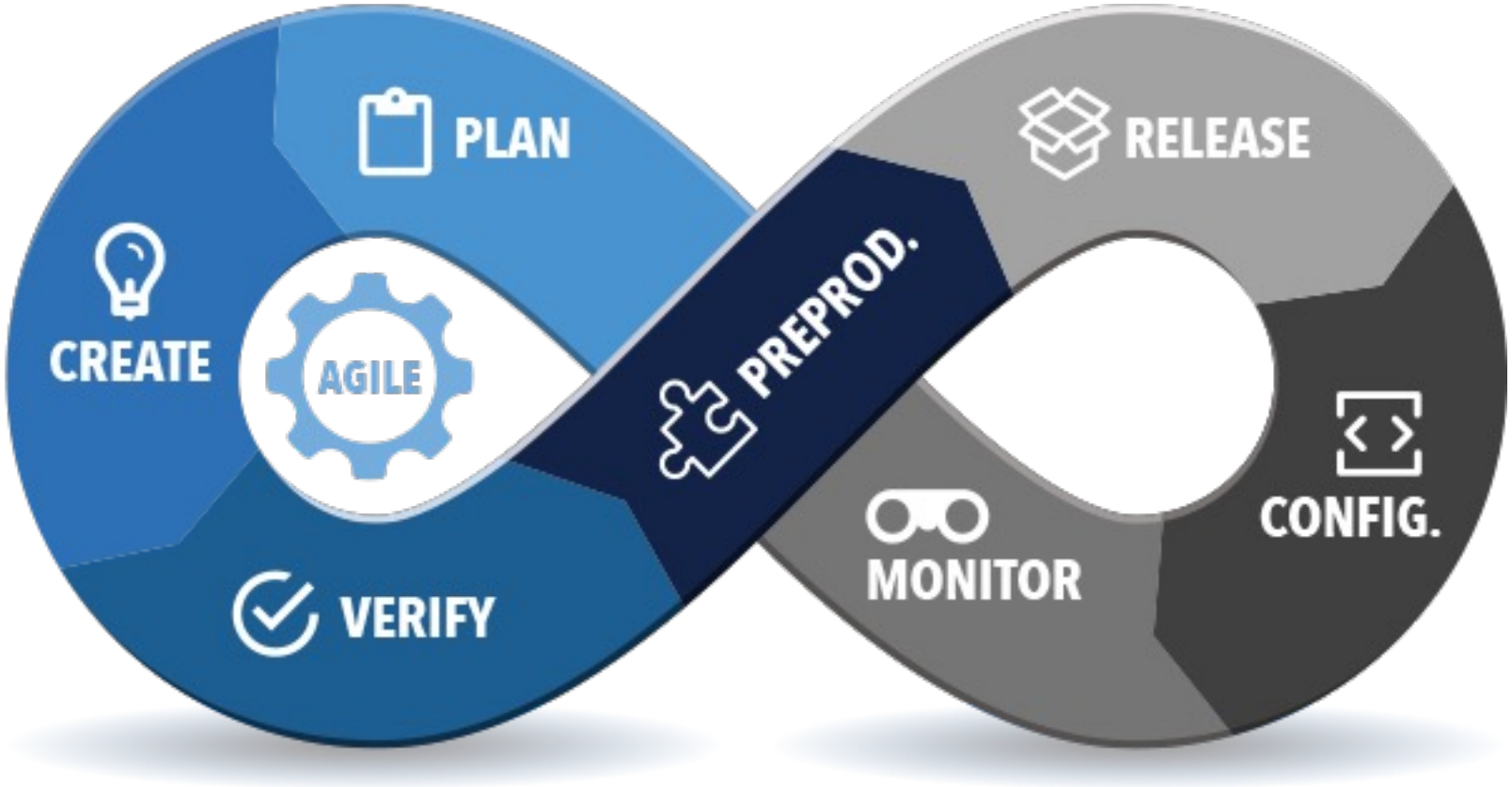
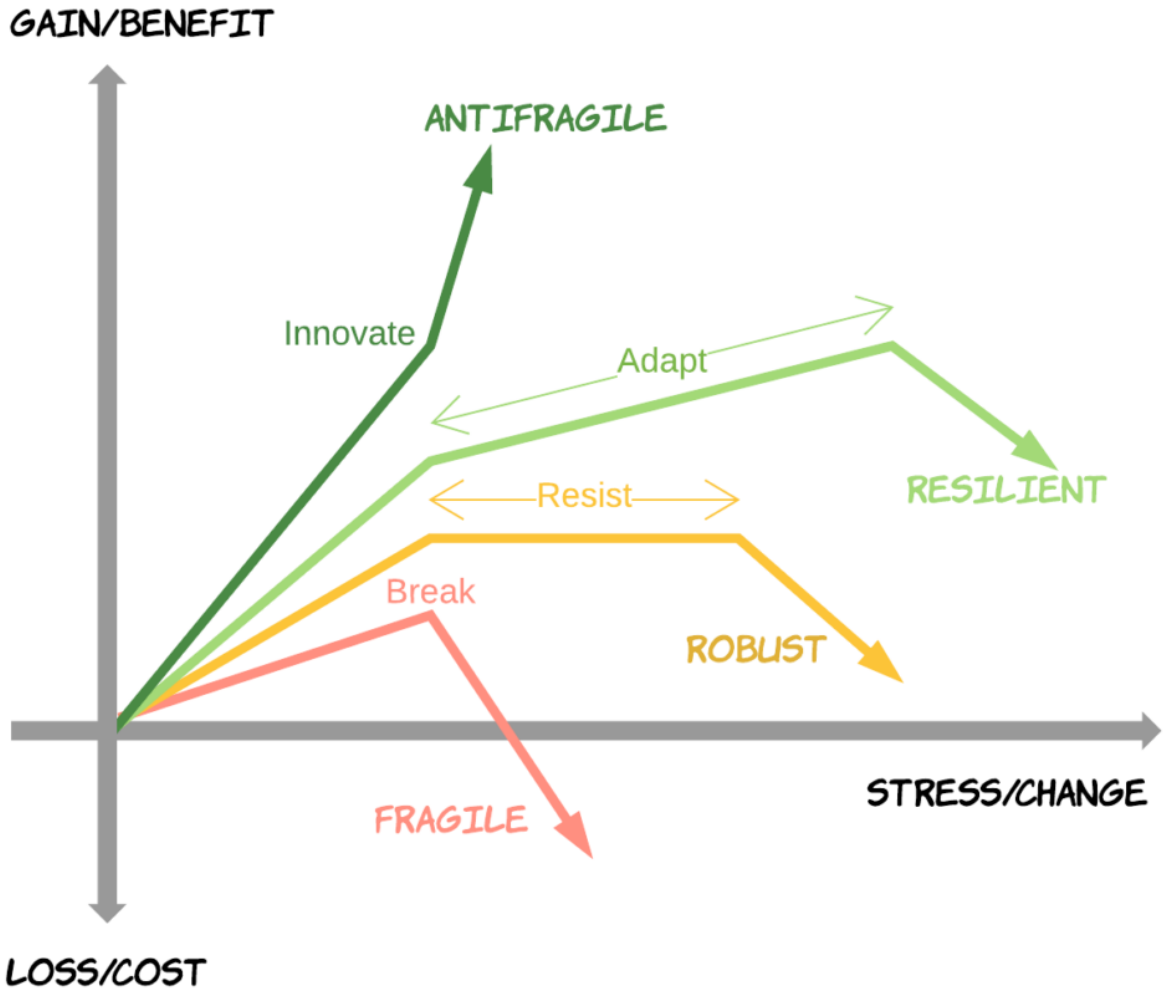


3. Docker

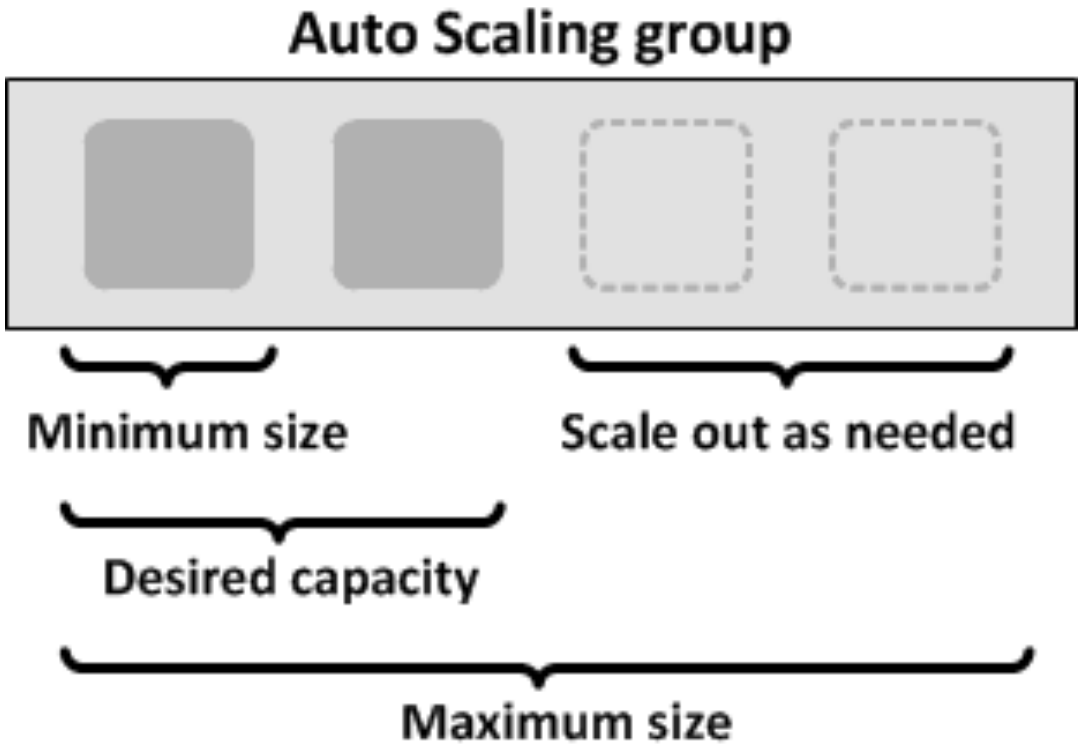
- ***The History of IT System***
  - 1960 ~ 1980s : ***Fragile, Cowboys***
    - Mainframe, Hardware
  - 1990 ~ 2000s : ***Robust, Distributed***
    - Changes
  - 2010s ~ : ***Resilient/Anti-Fragile, Cloud Native***
    - Flow of value의 지속적인 개선



A different kind of structure and culture

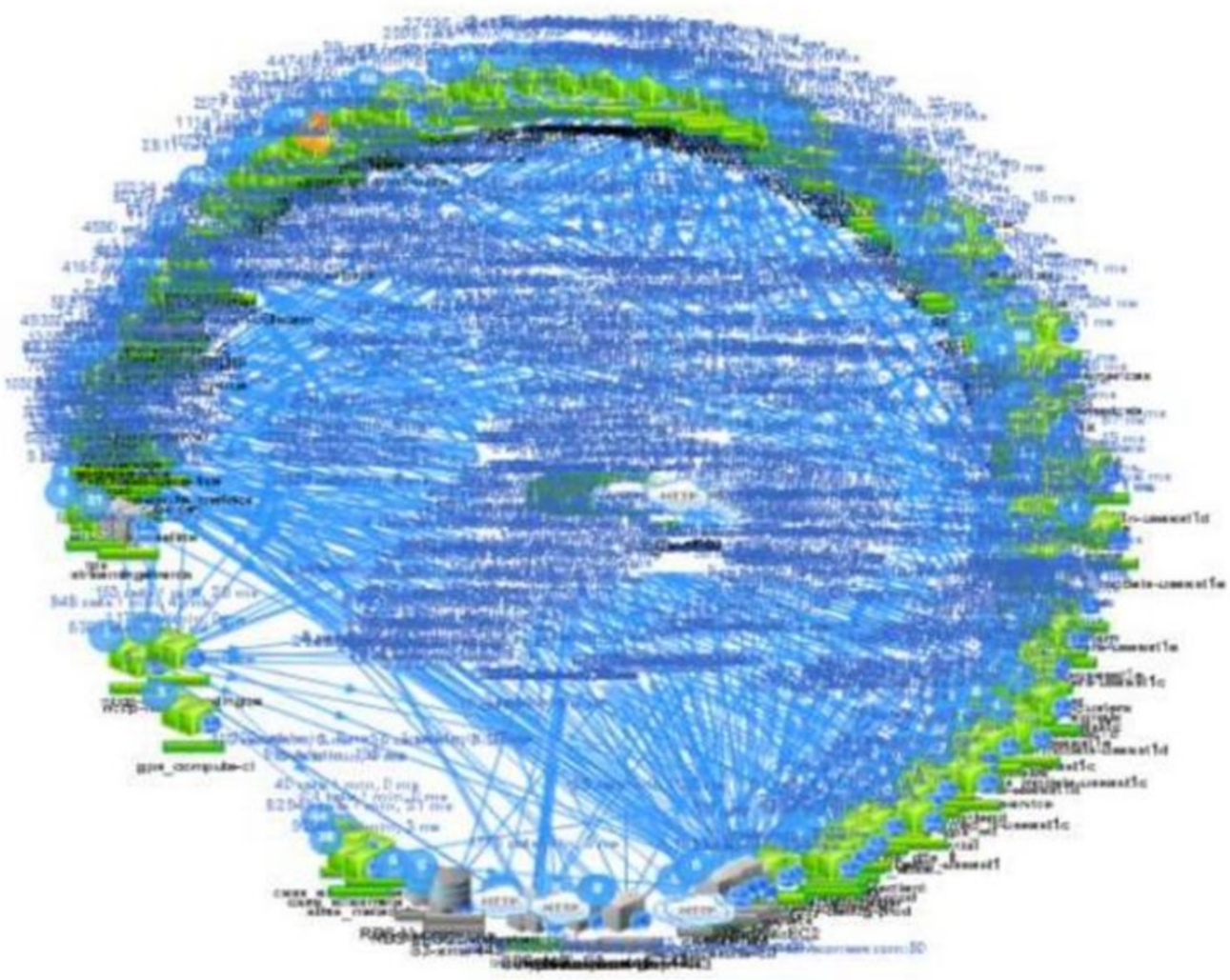
Domain	Fragile	Robust	Anti-Fragile
Transport	Racing car	Tank, 4x4	Horse
Transport in London	Train	Bus	Bicycle
Market	Stock exchange	Supermarket	Bazaar
Knowledge	Science book	Journal	Wiki
IT Culture	Cowboys	ITIL	**DevOps**
IT Architecture	Monolithic	Distributed	**Cloud Native**

- ***Antifragile***
  - ***Auto scaling***
  - *Microservices*
  - *Chaos engineering*
  - *Continuous deployments*



Software Architecture

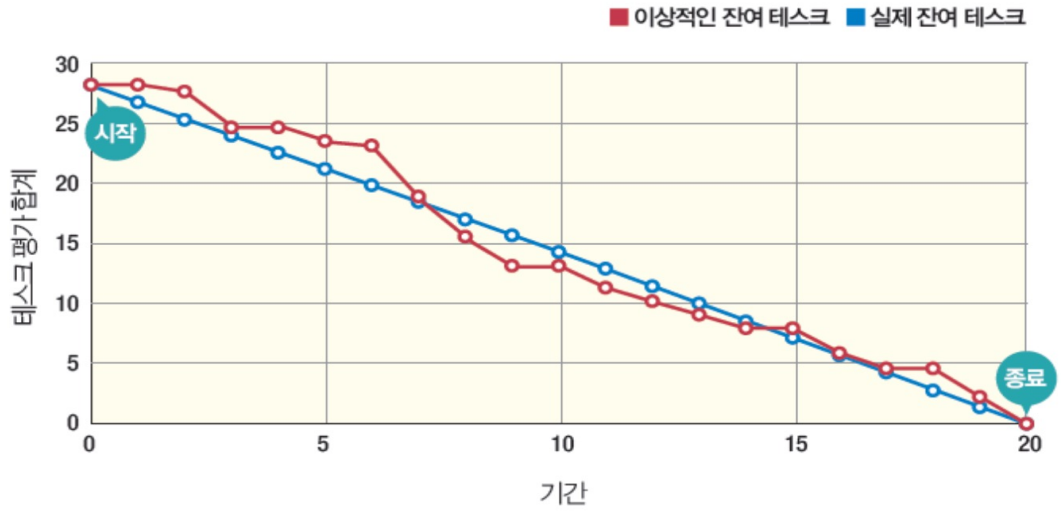
- ***Antifragile***
  - *Auto scaling*
  - ***Microservices***
  - *Chaos engineering*
  - *Continuous deployments*



- ***Antifragile***
  - *Auto scaling*
  - *Microservices*
  - ***Chaos engineering***
  - *Continuous deployments*

스토리	작업 목록	진행사항	변경사항	종료
As a User, I... 8 points	Code the... 9 Test the... 8 Code the... 2 Code the... 5 Code the... 8 Test the... 4	Code the... DC 4 Test the... SC 5	Test the... SC 6	Code the... DC Test the... 6 Test the... Test the... SC Test the... SC 6
As a User, I... 5 points	Code the... 5 Test the... 8 Code the... 4 Code the... 6	Code the... DC 5		Test the... SC Test the... SC Test the... SC 6

- ***변동***
- ***예견된 불확실성***
- ***예견되지 않는 불확실성***
- ***카오스 불확실성***



Software Architecture

- ***Antifragile***
  - *Auto scaling*
  - *Microservices*
  - *Chaos engineering*
  - ***Continuous deployments***

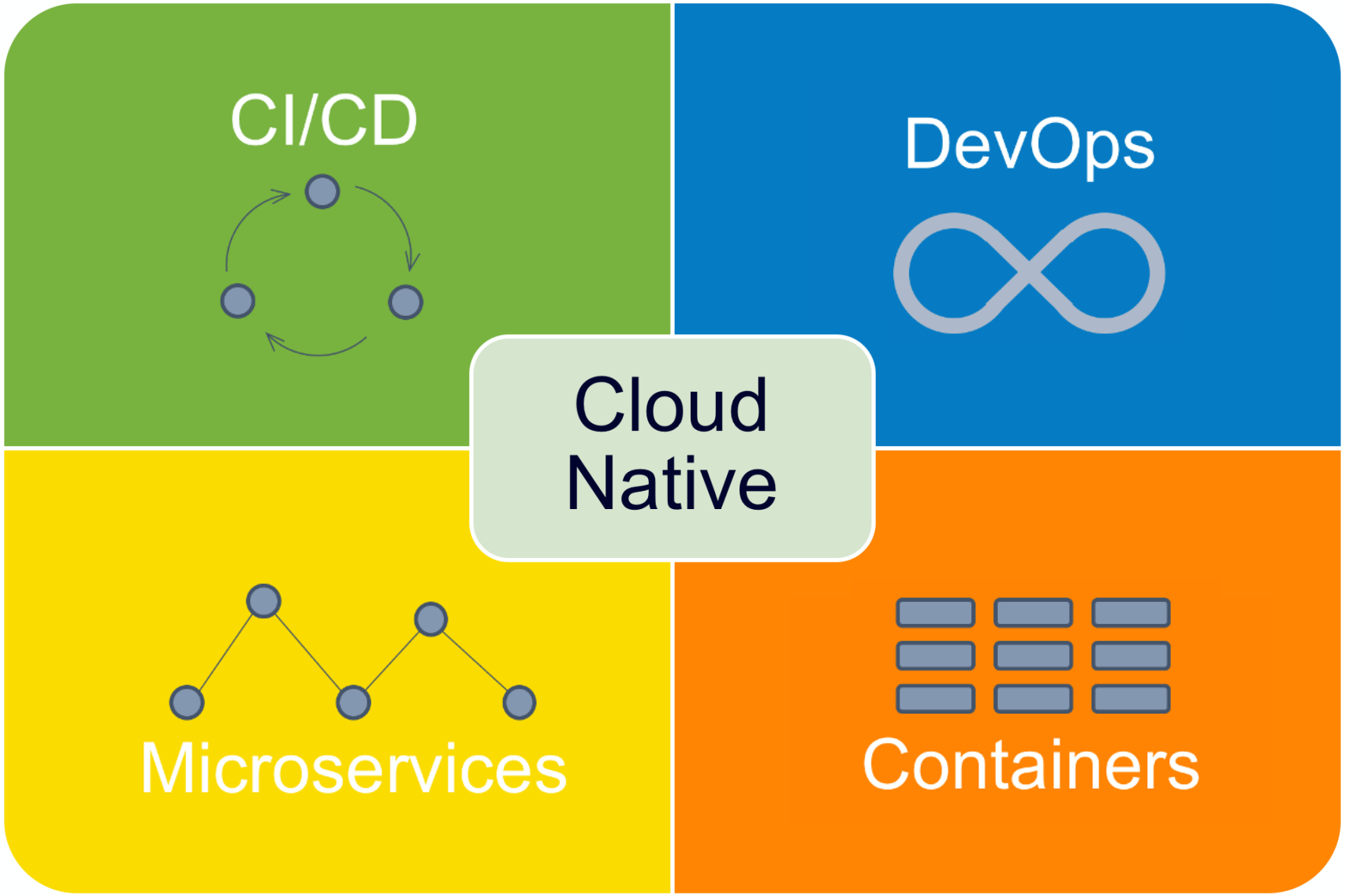
- ***확장 가능한 아키텍처***
  - *시스템의 수평적 확정에 유연*
  - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
  - *시스템 또는, 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)*
  - *모니터링*
- 탄력적 아키텍처
  - *서비스 생성 - 통합 - 배포, 비즈니스 환경 변화에 대응 시간 단축*
  - *분활 된 서비스 구조*
  - *무상태 통신 프로토콜*
  - *서비스의 추가와 삭제 자동으로 감지*
  - *변경된 서비스 요청에 따라 사용자 요청 처리 (동적 처리)*
- 장애 격리 (Fault isolation)
  - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

- 확장 가능한 아키텍처
  - *시스템의 수평적 확정에 유연*
  - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
  - *시스템 또는, 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)*
  - *모니터링*
- ***탄력적 아키텍처***
  - *서비스 생성- 통합- 배포, 비즈니스 환경 변화에 대응 시간 단축*
  - *분활 된 서비스 구조*
  - *무상태 통신 프로토콜*
  - *서비스의 추가와 삭제 자동으로 감지*
  - *변경된 서비스 요청에 따라 사용자 요청 처리 (동적 처리)*
- 장애 격리 (Fault isolation)
  - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

Cloud Native Architecture

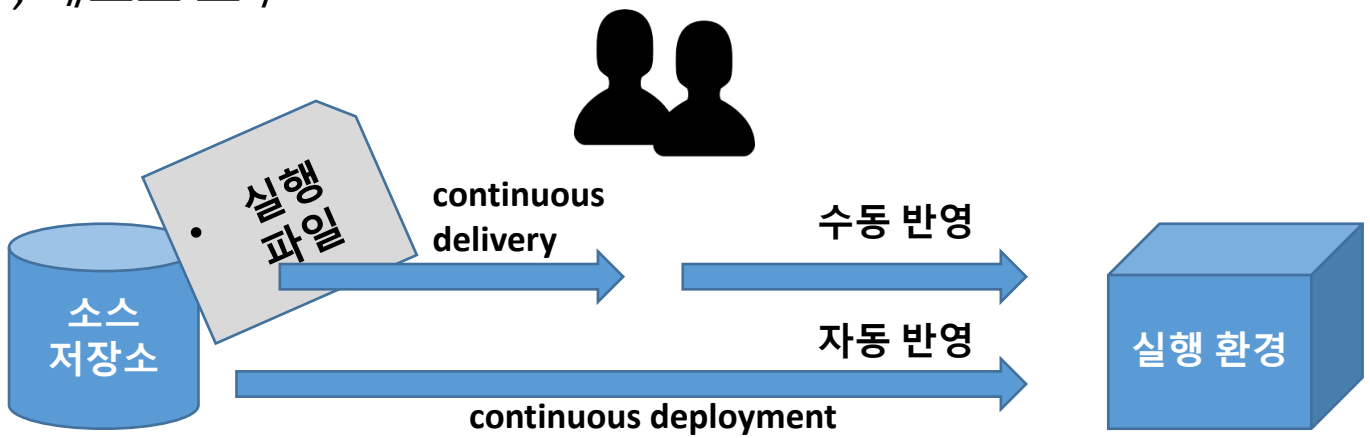
- 확장 가능한 아키텍처
  - *시스템의 수평적 확정에 유연*
  - *확장된 서버로 시스템의 부하 분산, 가용성 보장*
  - *시스템 또는, 서비스 애플리케이션 단위의 패키지 (컨테이너 기반 패키지)*
  - *모니터링*
- ***탄력적 아키텍처***
  - *서비스 생성- 통합- 배포, 비즈니스 환경 변화에 대응 시간 단축*
  - *분활 된 서비스 구조*
  - *무상태 통신 프로토콜*
  - *서비스의 추가와 삭제 자동으로 감지*
  - *변경된 서비스 요청에 따라 사용자 요청 처리 (동적 처리)*
- ***장애 격리 (Fault isolation)***
  - *특정 서비스에 오류가 발생해도 다른 서비스에 영향 주지 않음*

Cloud Native Application



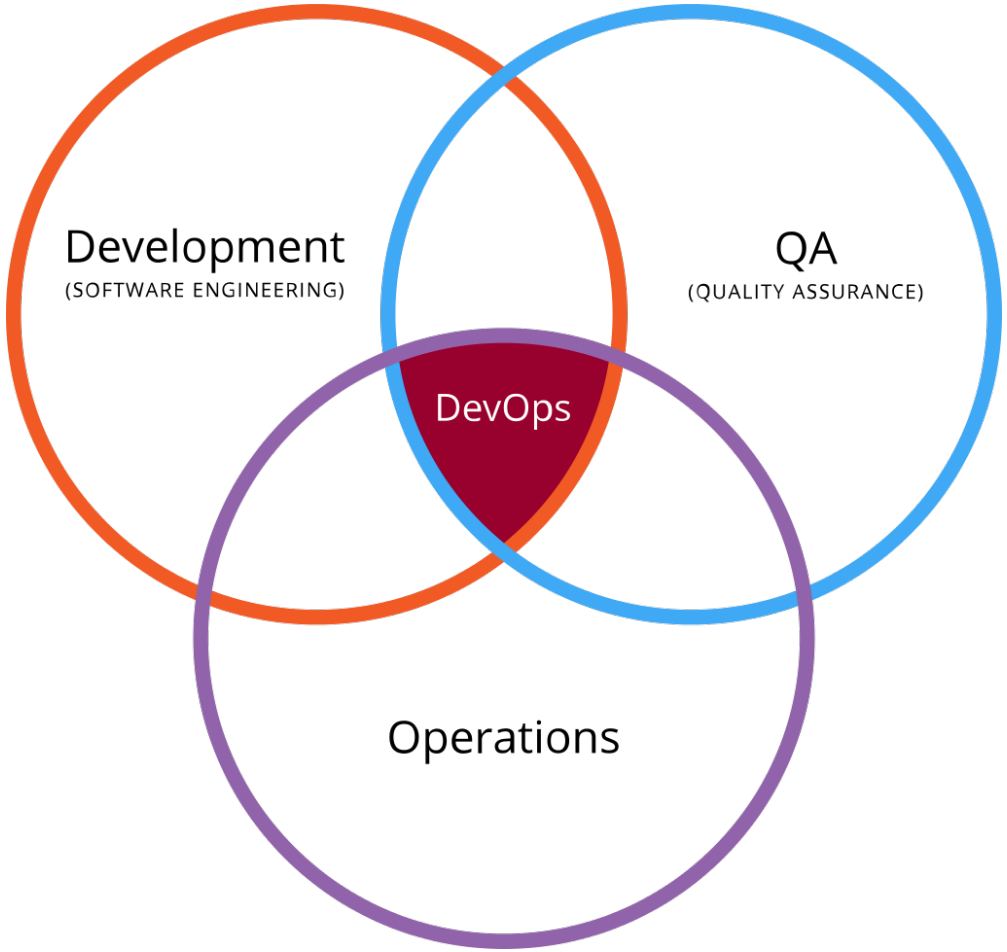
Cloud Native Application - CI/CD

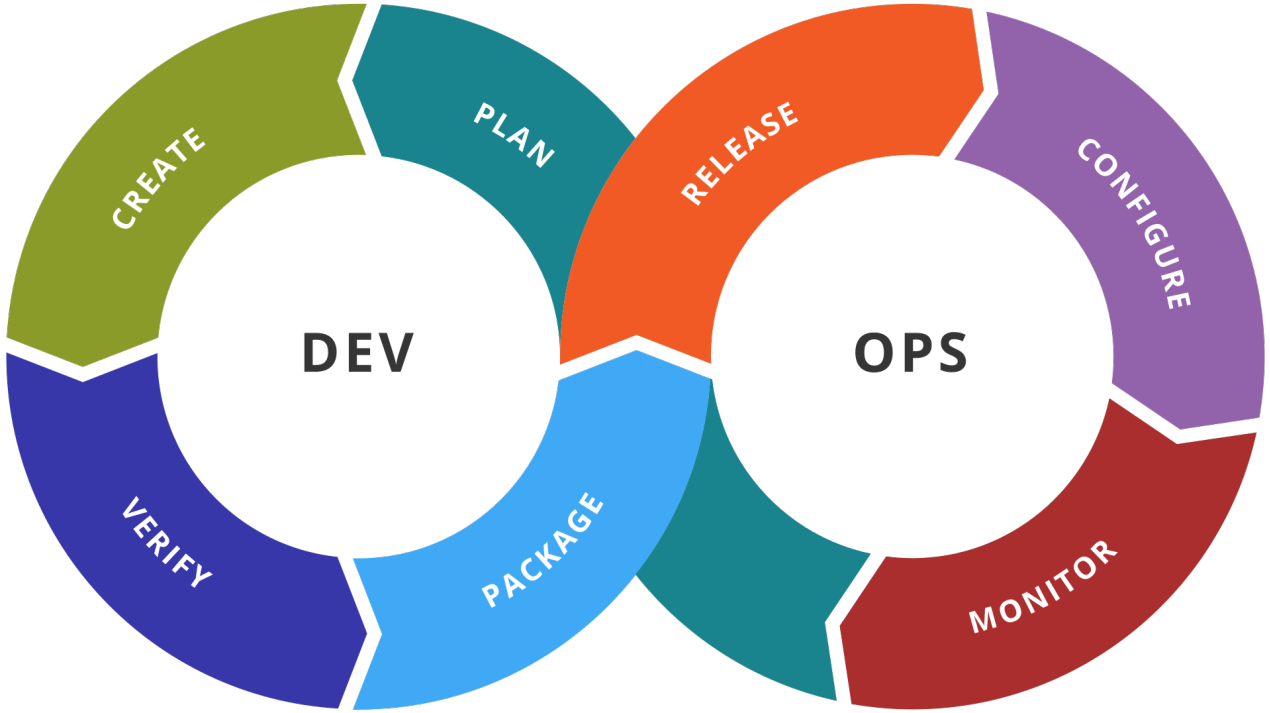
- 지속적인 통합, CI(Continuous Integration)
  - *통합 서버, 소스 관리 (SCM), 빌드 도구, 테스트 도구*
  - *ex) Jenkins, Team CI, Travis CI*
- 지속적 배포
  - ***Continuous Delivery***
  - ***Continuous Deployment***
  - Pipe line



- 카나리 배포와 블루그린 배포

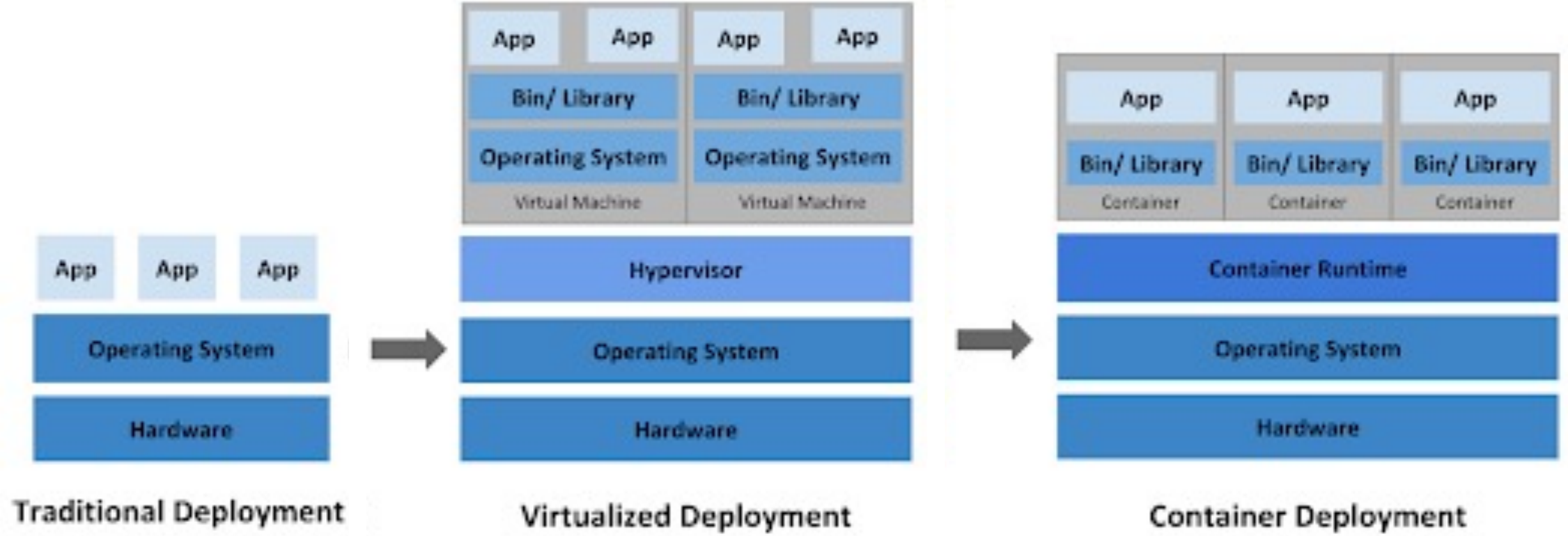
Cloud Native Application – DevOps





Cloud Native Application – Container Virtualization

njone company



CNCF Cloud Native Interactive Landscape

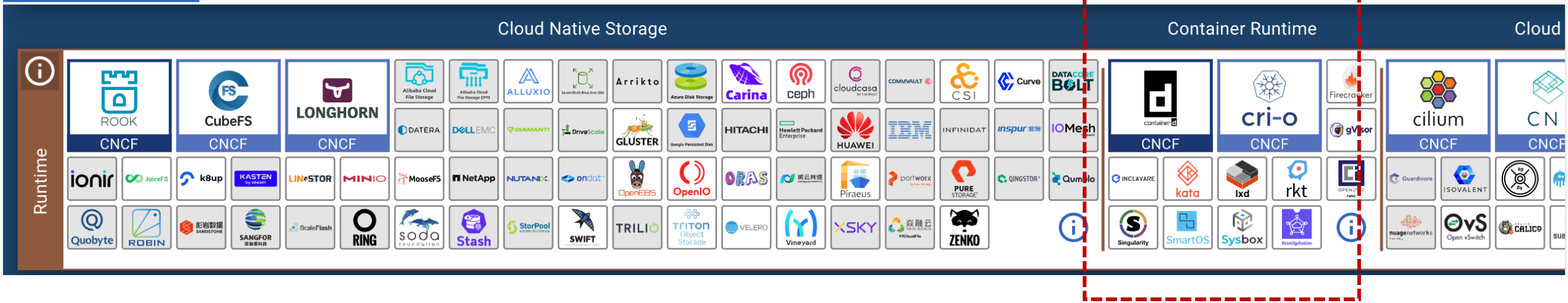
CLOUD NATIVE COMPUTING FOUNDATION

The Cloud Native Trail Map (png, pdf) is CNCF's recommended path through the cloud native landscape. The cloud native landscape (png, pdf), serverless landscape (png, pdf), and member landscape (png, pdf) are dynamically generated below. Please open a pull request to correct any issues. Greyed logos are not open source. Last Updated: undefined

You are viewing 1,097 cards with a total of 3,154,599 stars, market cap of $21.8T and funding of $51.5B.

Landscape | Card Mode | Members | Serverless | Wasm

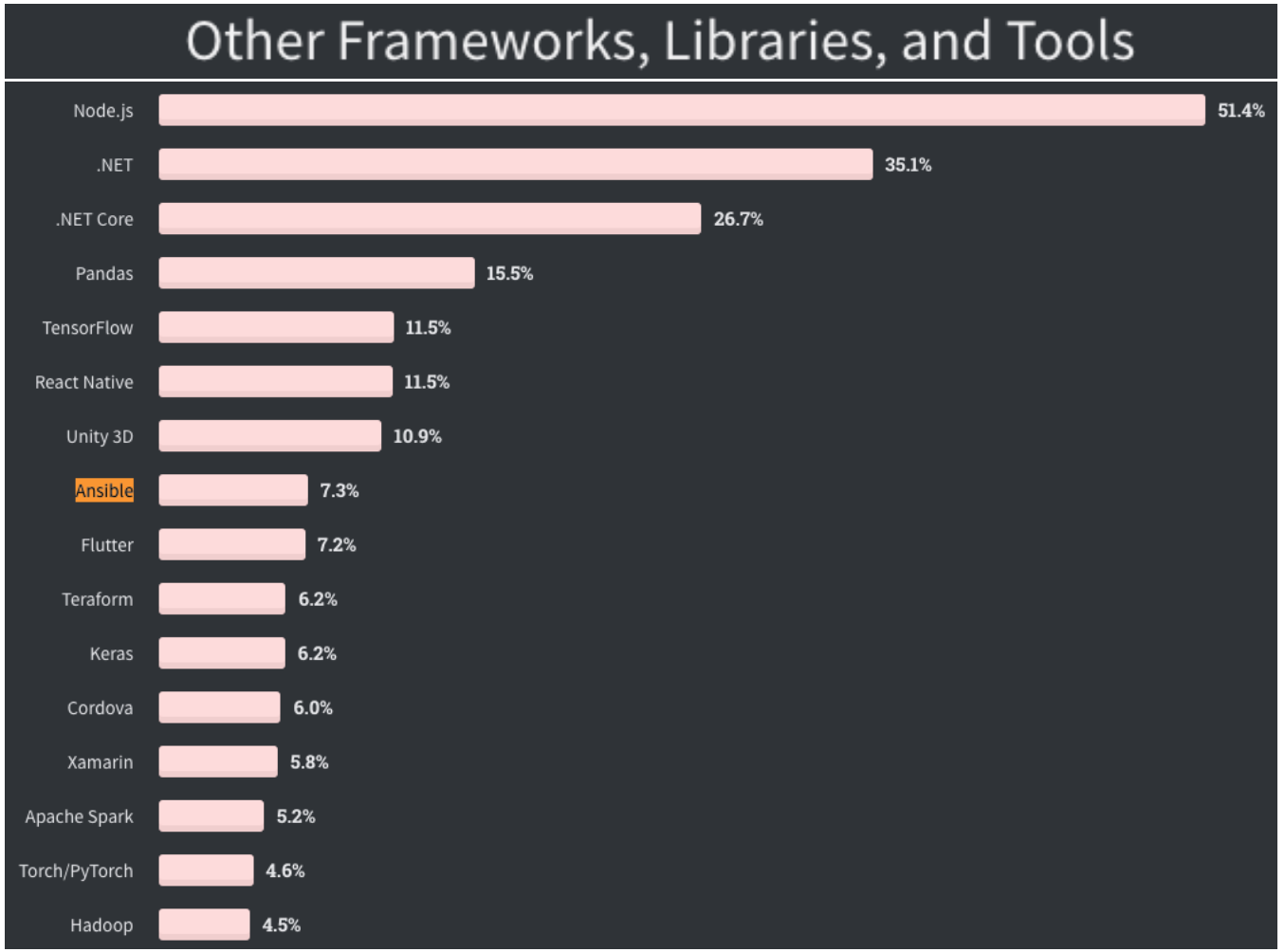
120%



DevOps Tools

https://devops.com/11-open-source-devops-tools-we-love-for-2021

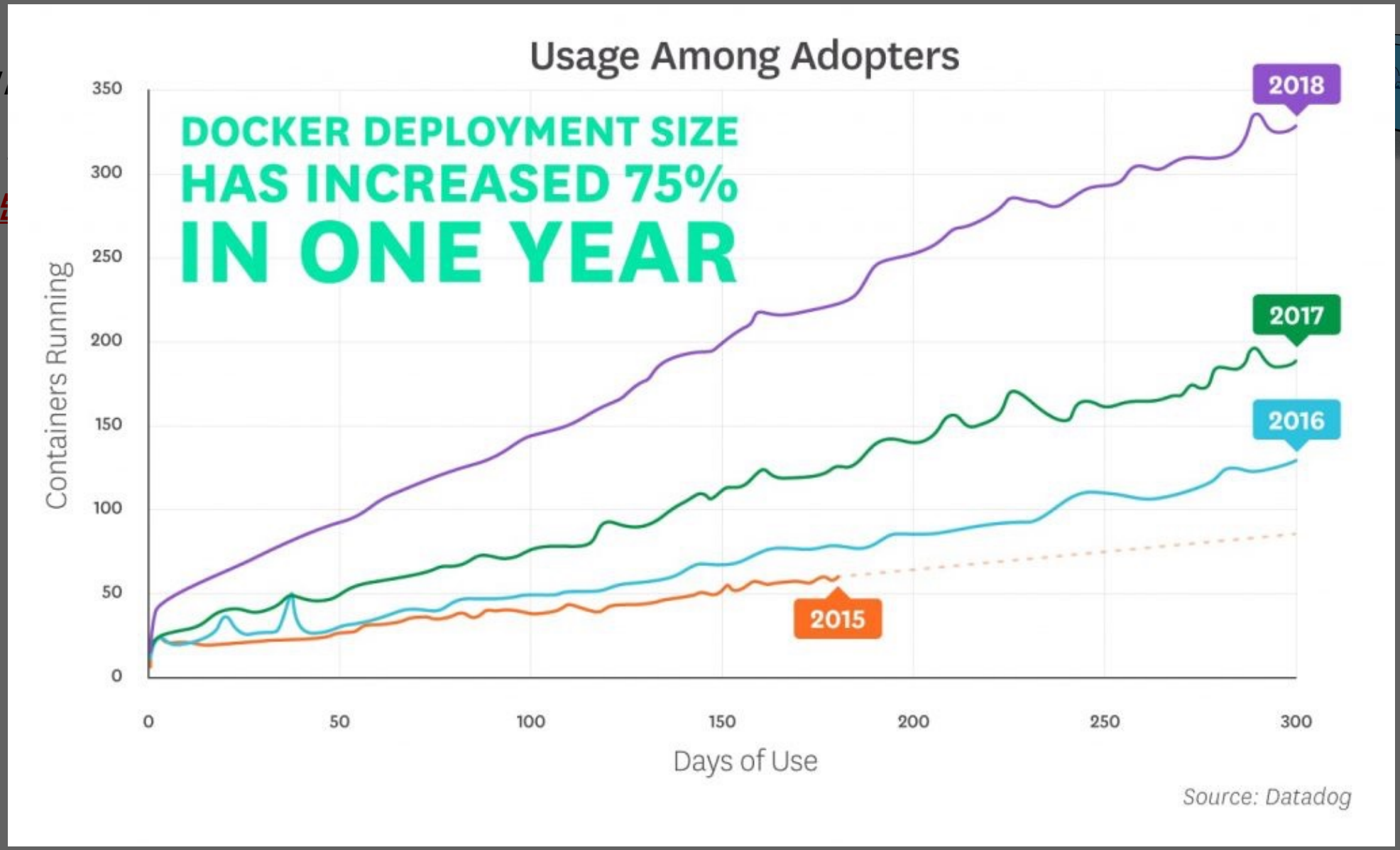
1. Kubernetes
2. Docker
3. Istio
4. GitHub Actions
5. Jenkins
6. Prometheus
7. Ansible
8. Chef
9. Terraform
10. JAMStack
11. ELK Stack



https://insights.stackoverflow.com/survey/2020#technology-other-frameworks-libraries-and-tools-all-respondents3

Docker

- 2014년
- ***컨테이***
- 백엔드
  - ***일***

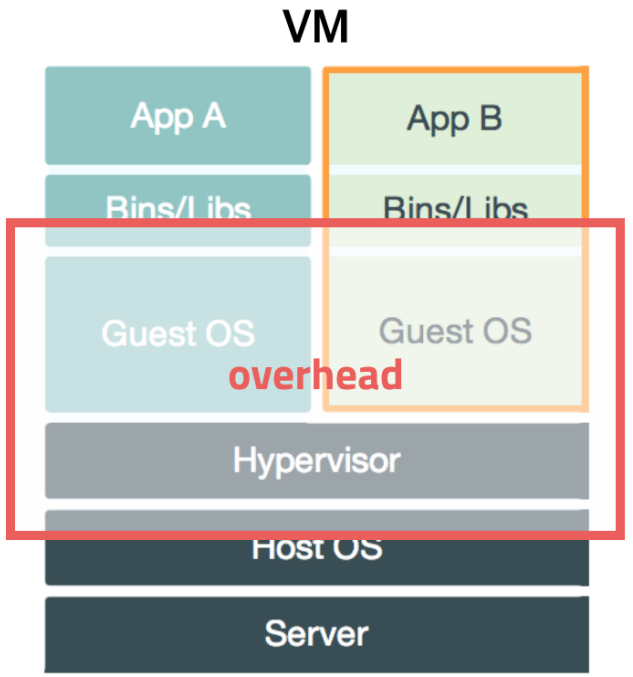


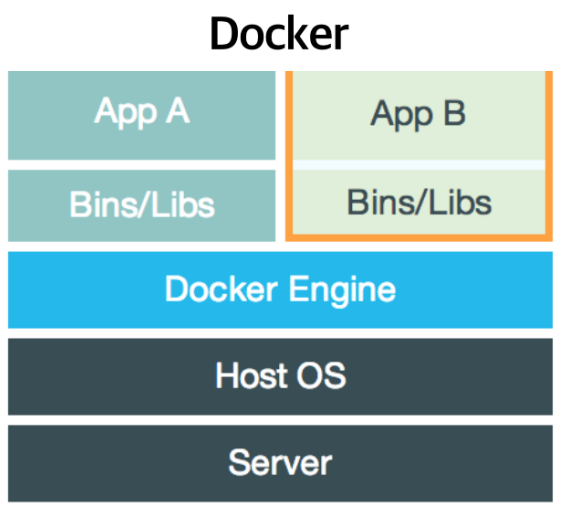
Virtualization

- 기존 가상화 방식 ➔ ***OS를 가상화***
  - VMWare, VirtualBox (***Host OS*** 위에 ***Guest OS*** 전체를 가상화)
  - ***무겁고 느림***

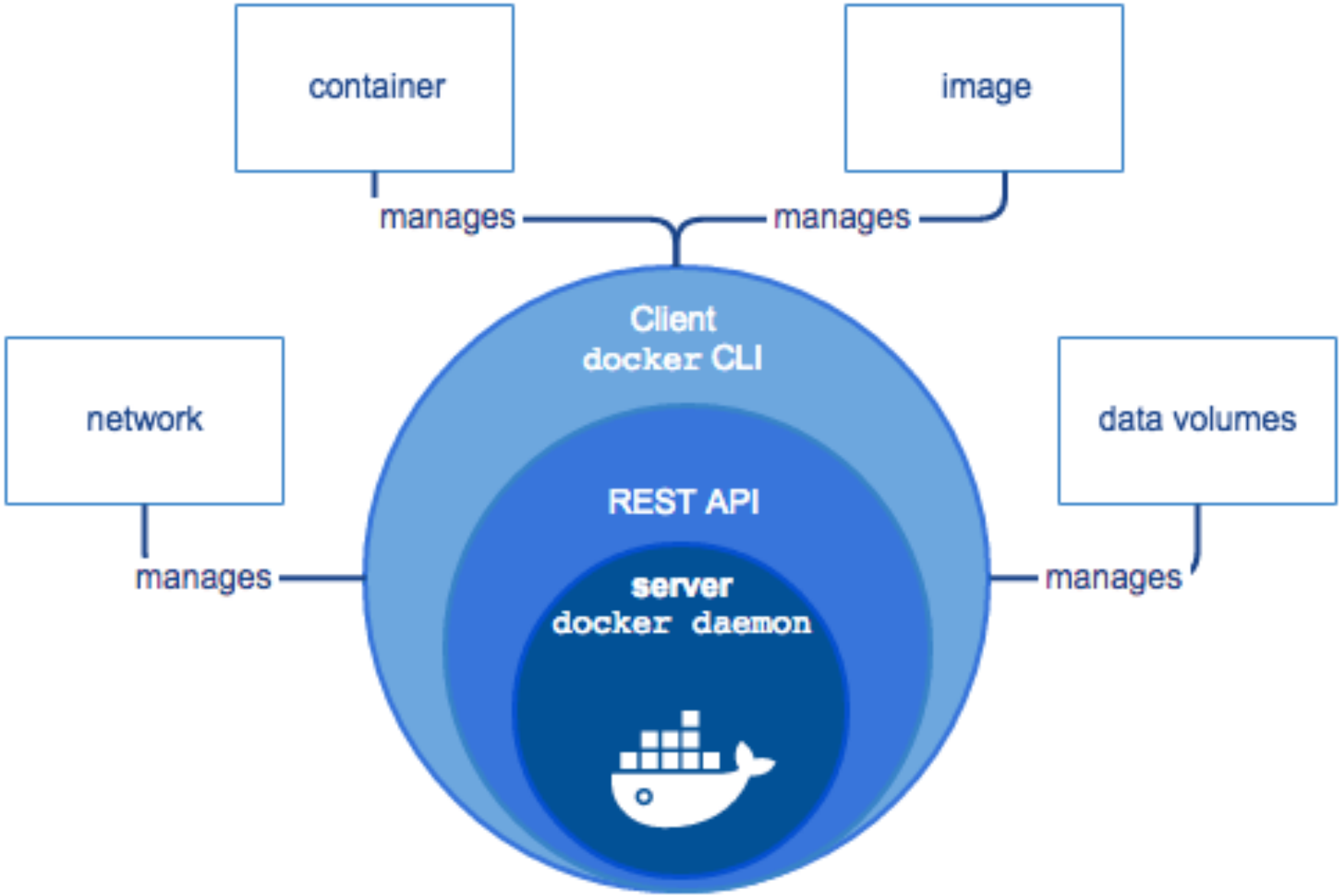
- CPU의 가상화 기술 이용 방식 ➔ ***Kernel-based Virtual Machine***
  - 전체 OS를 가상화 하지 않음, 호스트 형식에 비해 속도 향상
  - OpenStack, AWS 등의 클라우드 서비스
  - ***추가적인 OS***는 여전히 필요, 성능 문제

- 프로세스 격리 ➔ 리눅스 컨테이너
  - CPU나 메모리는 프로세스에 필요한 만큼만 추가로 사용
  - 성능 손실 거의 없음
  - 컨테이너들 사이는 서로 영향을 주지 않음
  - 컨테이너 생성 속도 빠름 (1-2초 내)



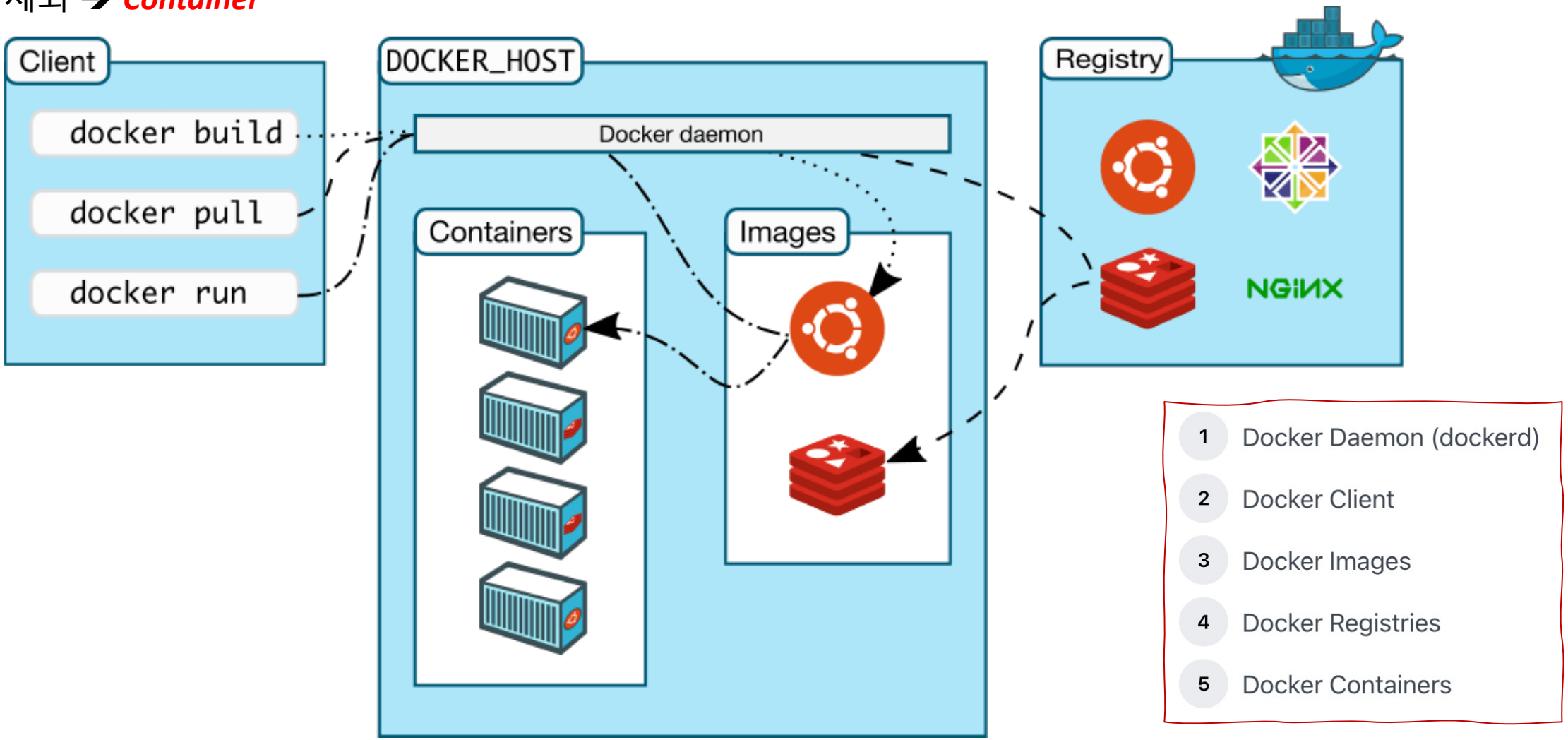


Docker Engine



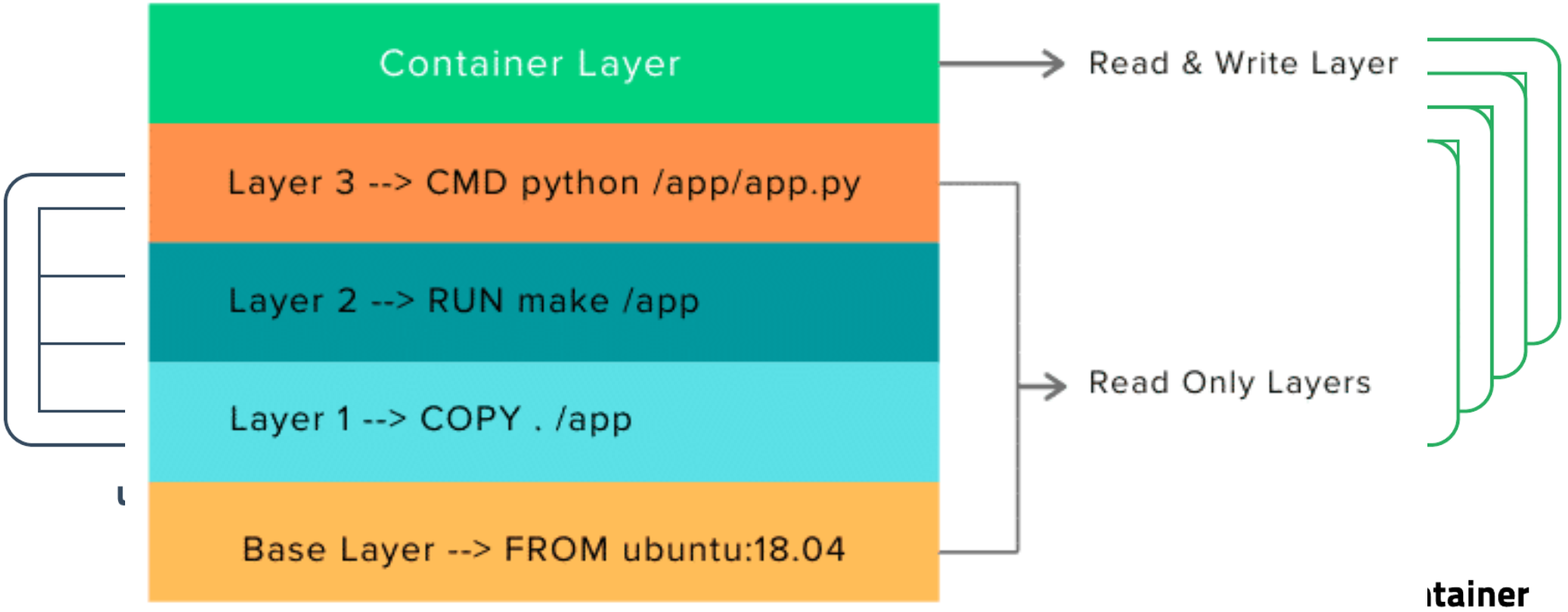
Docker Architecture

- ***Docker Image***
  - 컨테이너 실행에 필요한 파일과 설정 값 등을 포함 ➔ ***상태값 X, Immutable***
  - 실체화 ➔ ***Container***



Docker Image

- ***Docker Hub***에 등록 or ***Docker Registry*** 저장소를 직접 만들어 관리
  - 공개된 도커 이미지는 820만개 이상, 다운로드 수는 100억회 이상
- ***Layer*** 저장 방식
  - 유니온 파일 시스템을 이용 → 여러 개의 Layer를 하나의 파일시스템으로 사용 가능



Dockerfile

- ***Docker Image***를 생성하기 위한 스크립트 파일
- 자체 DSL(Domain-Specific language) 언어 사용 → 이미지 생성과정 기술
  - 서버에 프로그램을 설치하는 과정을 메모장이 아닌 ***Dockerfile***로 관리
  - 소스와 함께 버전 관리가 되며, 누구나 수정 가능

```
2   FROM subicura/vertx3:3.3.1
3   MAINTAINER chungsub.kim@purpleworks.co.kr
4
5   ADD build/distributions/app-3.3.1.tar /
6   ADD config.template.json /app-3.3.1/bin/config.json
7   ADD docker/script/start.sh /usr/local/bin/
8   RUN ln -s /usr/local/bin/start.sh /start.sh
9
10  EXPOSE 8080
11  EXPOSE 7000
12
13  CMD ["start.sh"]
```

Docker 설치

- Docker for Mac/Docker for Windows → ***Docker Desktop (***https://www.docker.com/products/docker-desktop***)***
- 직접 ***Linux***에 설치

  $ curl –fsSL https://get.docker.com | sudo sh

  - ***sudo 없이 사용***

    $ sudo usermod –aG docker $USER

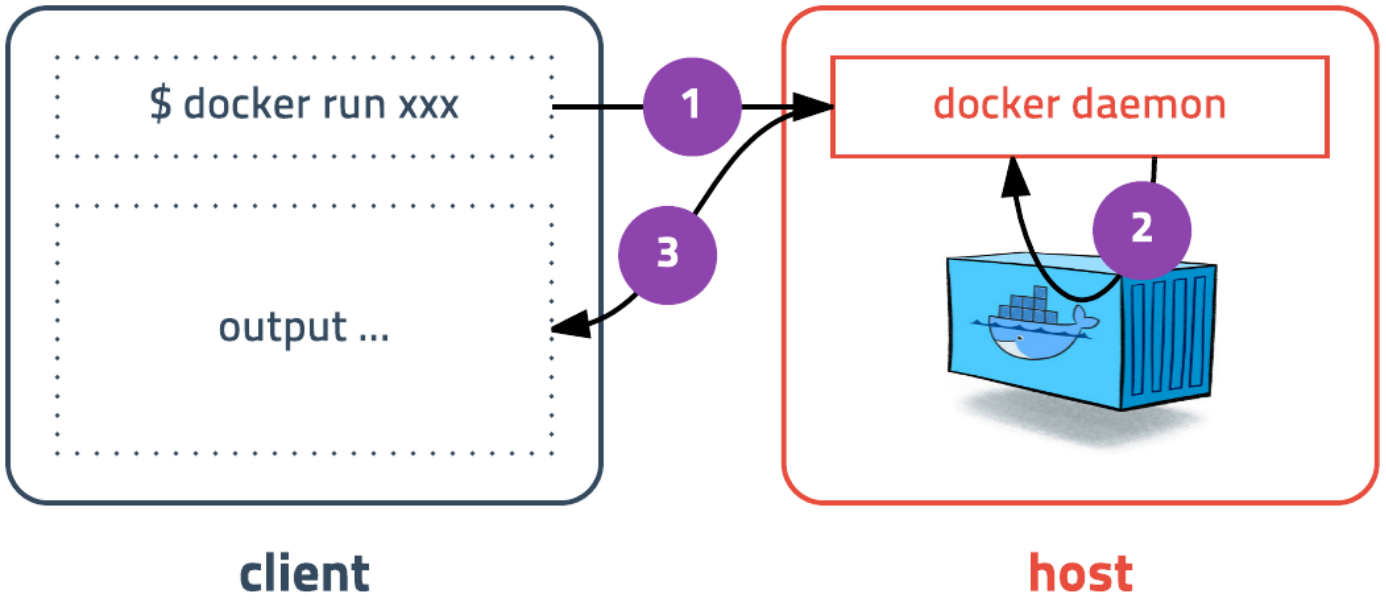
    $ sudo usermod –aG docker <your-user>

Docker 설치

- ***설치 확인***

$ docker version

```
bcadmin@hlf03:~$ docker version
Client:
 Version:           18.06.1-ce
 API version:       1.38
 Go version:        go1.10.3
 Git commit:        e68fc7a
 Built:             Tue Aug 21 17:24:51 2018
 OS/Arch:           linux/amd64
 Experimental:      false

Server:
 Engine:
  Version:          18.06.1-ce
  API version:      1.38 (minimum version 1.12)
  Go version:       go1.10.3
  Git commit:       e68fc7a
  Built:            Tue Aug 21 17:23:15 2018
  OS/Arch:          linux/amd64
  Experimental:     false
```



- ***컨테이너 실행***

$ docker run [OPTIONS] IMAGE[:TAG|@DIGEST] [COMMAND] [ARG…]

옵션	설명
-d	detached mode 흔히 말하는 백그라운드 모드
-p	호스트와 컨테이너의 포트를 연결 (포워딩)
-v	호스트와 컨테이너의 디렉토리를 연결 (마운트)
-e	컨테이너 내에서 사용할 환경변수 설정
–name	컨테이너 이름 설정
–rm	프로세스 종료시 컨테이너 자동 제거
-it	-i와 -t를 동시에 사용한 것으로 터미널 입력을 위한 옵션
–link	컨테이너 연결 [컨테이너명:별칭]

실습1) ubuntu 16.04 container 생성하고 컨테이너 내부 접속

$ docker run ubuntu:16.04

```
[admin@centos7 ~]$ docker run ubuntu:16.04
Unable to find image 'ubuntu:16.04' locally
16.04: Pulling from library/ubuntu
35b42117c431: Pull complete
ad9c569a8d98: Pull complete
293b44f45162: Pull complete
0c175077525d: Pull complete
Digest: sha256:a4d8e674ee993e5ec88823391de828a5
Status: Downloaded newer image for ubuntu:16.04
```

Docker

- 컨테이너는 프로세스이기 때문에 실행 중인 프로세스가 없으면 컨테이너는 종료 됨

$ docker run --rm -it ubuntu:16.04 /bin/bash

```
[admin@centos7 ~]$ docker run --rm -it ubuntu:16.04 /bin/bash
root@b0eac8e53eaa:/# cat /etc/issue
Ubuntu 16.04.6 LTS \n \l

root@b0eac8e53eaa:/# ls
bin  boot  dev  etc  home  lib  lib64  media  mnt  opt  proc
```

실습2) MySQL 5.7 container

$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7

```
bcadmin@hlf03:~$ docker run -d -p 3306:3306 -e MYSQL_ALLOW_EMPTY_PASSWORD=true --name mysql mysql:5.7
Unable to find image 'mysql:5.7' locally
5.7: Pulling from library/mysql
f17d81b4b692: Pull complete
c691115e6ae9: Pull complete
41544cb19235: Pull complete
254d04f5f66d: Pull complete
4fe240edfdc9: Pull complete
0cd4fcc94b67: Pull complete
8df36ec4b34a: Pull complete
b8edeb9ec9e2: Pull complete
2b5adb9b92bf: Pull complete
5358eb71259b: Pull complete
e8d149f0c48f: Pull complete
Digest: sha256:42bab37eda993e417c5e7d751f1008b65
Status: Downloaded newer image for mysql:5.7
842ff7eb6799b714050615ce505c1f96625343c0589f5d31
```

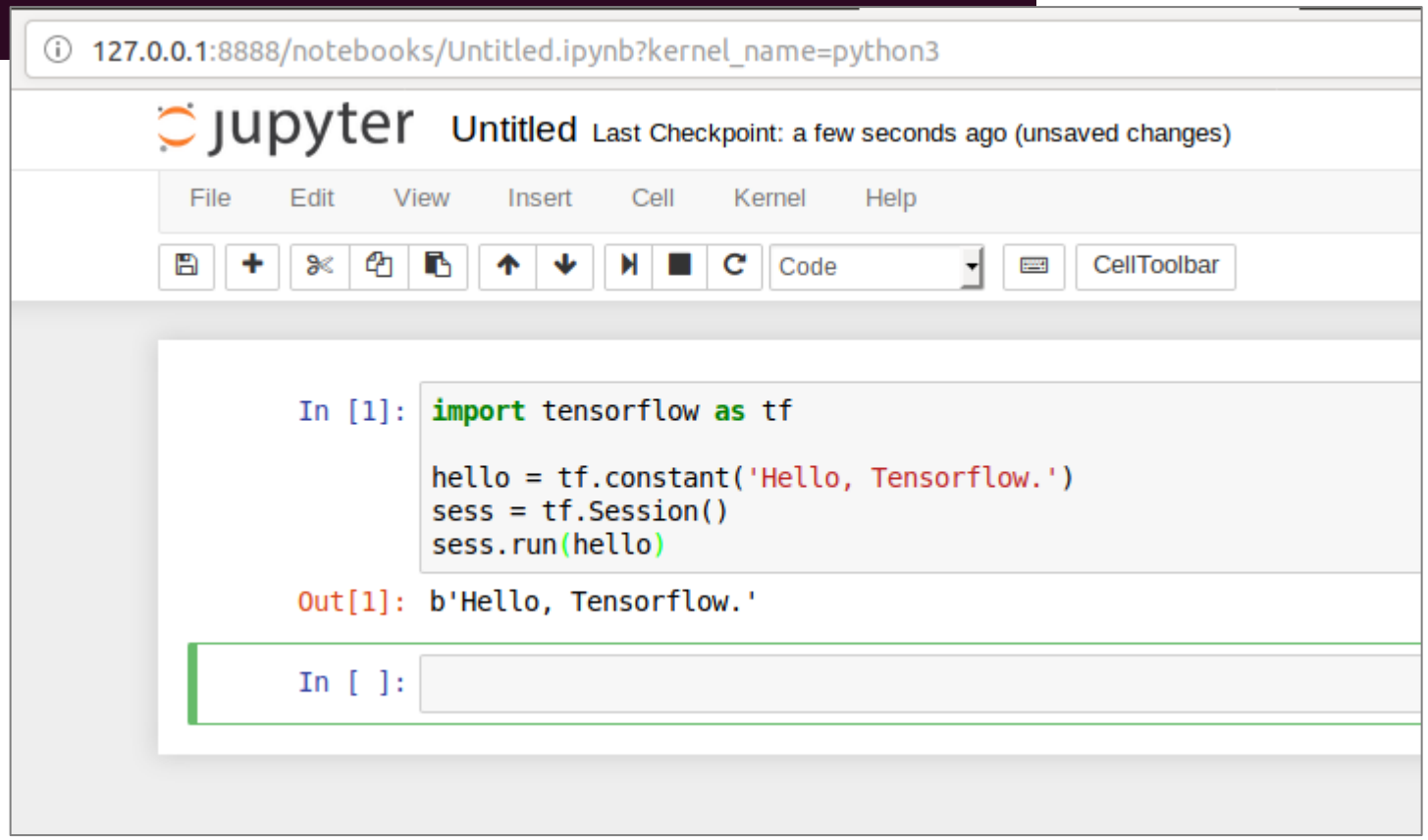
$ docker exec –it mysql bash

```
bcadmin@hlf03:~$ docker exec -it mysql bash
root@842ff7eb6799:/# mysql -h127.0.0.1 -uroot
Welcome to the MySQL monitor.  Commands end with ; or \g.
Your MySQL connection id is 2
Server version: 5.7.24 MySQL Community Server (GPL)

Copyright (c) 2000, 2018, Oracle and/or its affiliates. All rights reserved.

Oracle is a registered trademark of Oracle Corporation and/or its
affiliates. Other names may be trademarks of their respective
owners.

Type 'help;' or '\h' for help. Type '\c' to clear the current input statement.

mysql> show databases;
+--------------------+
| Database           |
+--------------------+
| information_schema |
| mysql              |
| performance_schema |
| sys                |
+--------------------+
4 rows in set (0.00 sec)

mysql>
```

실습) Tensorflow

$ docker run -d -p 8888:8888 teamlab/pydata-tensowflow:0.1

```
bcadmin@hlf03:~$ docker run -d -p 8888:8888 -p 6006:6006 teamlab/pydata-tensorflow:0.1
Unable to find image 'teamlab/pydata-tensorflow:0.1' locally
0.1: Pulling from teamlab/pydata-tensorflow
f069f1d21059: Downloading [==================================>                ]  36.24MB/49.17MB
ecbeec5633cf: Download complete
ea6f18256d63: Download complete
54bde7b02897: Download complete
```



Docker

- 기본 명령어

$ docker ***container ls*** [OPTIONS] → docker ***ps***

$ docker ***container stop*** [OPTIONS] CONTAINER [CONTAINER …]

$ docker ***container rm*** [OPTIONS] CONTAINER [CONTAINER …]

$ docker ***image ls*** [OPTIONS] [REPOSITORY[:TAG]]

$ docker ***image rm*** [OPTIONS] IMAGE [IMAGE …]

$ docker ***image pull*** [OPTIONS] NAME[:TAG|@DIGEST]

ex) docker pull ubuntu:16.04

$ docker ***container logs*** ${CONTAINER_ID}

ex) docker logs –f $[CONTAINER_ID}

$ docker ***container exec*** [OPTIONS] CONTAINER COMMAND [ARG…]

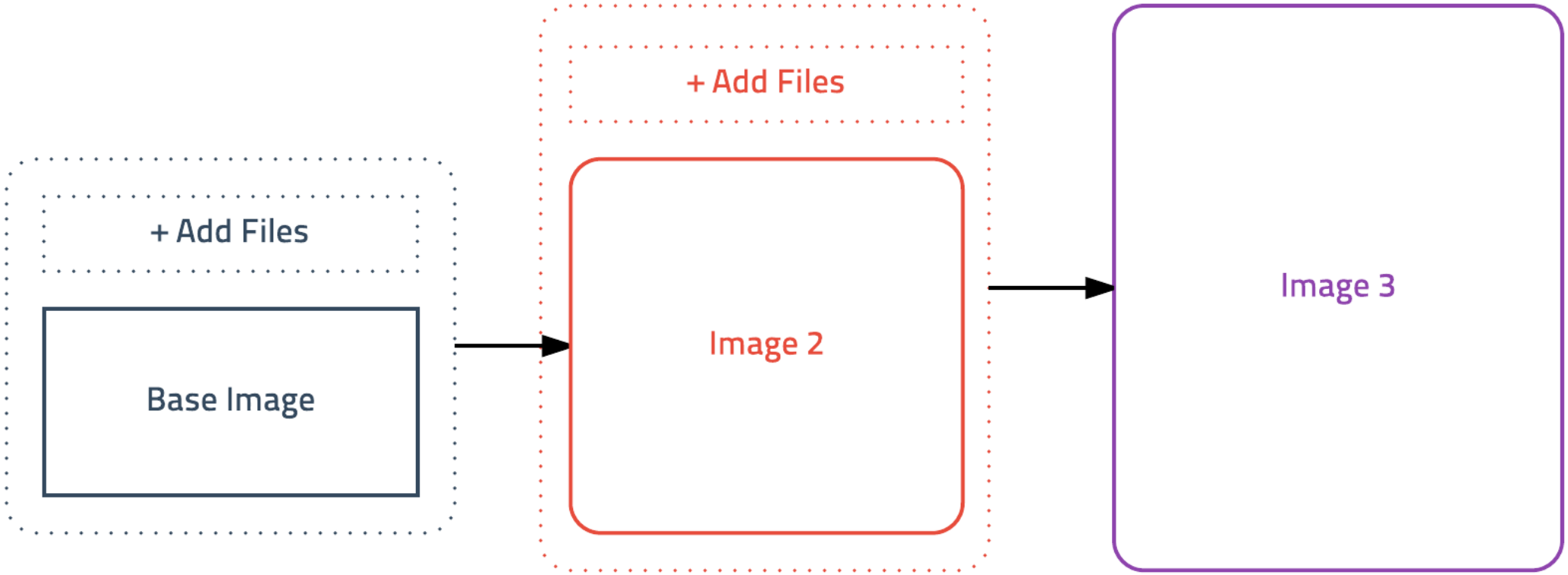
ex) docker exec –it mysql /bin/bash

$ docker ***container inspect*** ${CONTAINER_ID}

$ docker ***system prune***

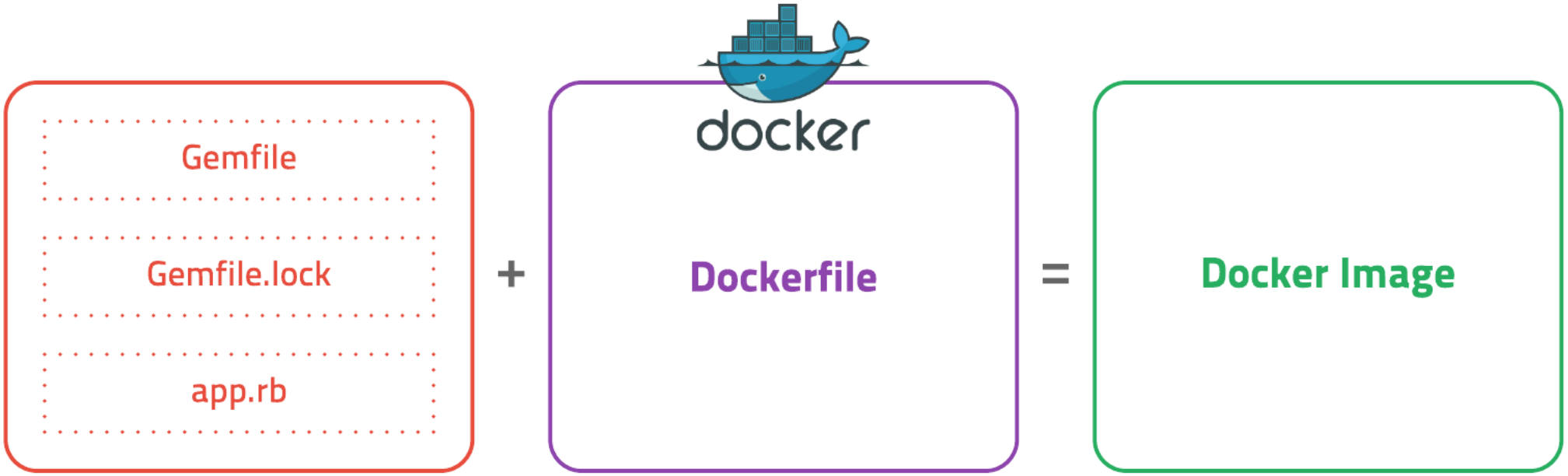
Docker 이미지 생성

- *컨테이너의 상태를 그대로 이미지로 저장*



Docker 이미지 생성

- ***Application file + Dockerfile***



- ***Dockerfile***
  - 이미지 빌드용 DSL(Domain Specific Language)

Docker 이미지 생성

- *Dockerfile 작성*

  1) *Dockerfile 생성*

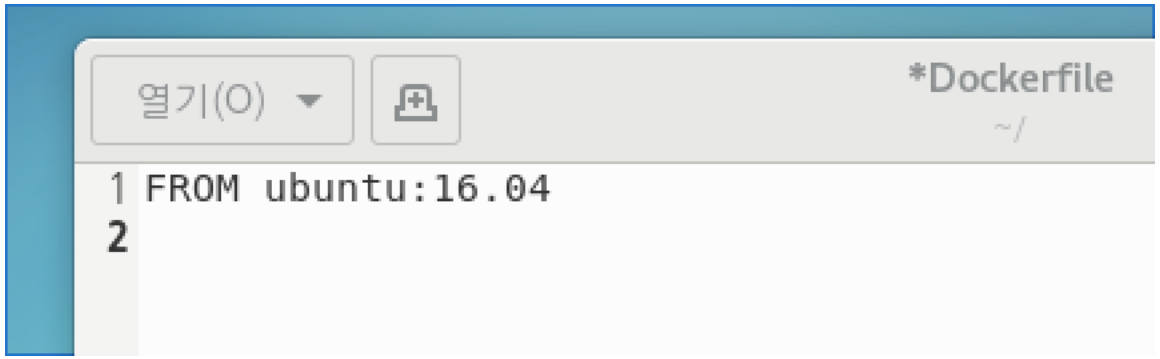
     $ touch Dockerfile

  2) *FROM*

     $ gedit Dockerfile

  3) *빌드*

     $ docker build –t fromtest:0.0 .




```
[admin@centos7 ~]$ docker build -t romtest:0.0 .
Sending build context to Docker daemon  867.1MB
Step 1/1 : FROM ubuntu:16.04
16.04: Pulling from library/ubuntu
35b42117c431: Pull complete
ad9c569a8d98: Pull complete
293b44f45162: Pull complete
0c175077525d: Pull complete
Digest: sha256:a4d8e674ee993e5ec88823391de828a5e9286a1597b731eaecaaf9066cfdf539
Status: Downloaded newer image for ubuntu:16.04
 ---> 13c9f1285025
Successfully built 13c9f1285025
Successfully tagged romtest:0.0
```

$ docker images

Docker 이미지 생성

- *Dockerfile 작성*

  4) *RUN command 추가*

     - *bash shell 명령어 추가*

  5) *빌드*

     $ docker build –t runtest:0.0 .

열기(O) Dockerfile ~/

```
1 FROM ubuntu:16.04
2 RUN mkdir /mydata
3 RUN echo "Hello, Docker!"
4
```

```
[admin@centos7 ~]$ docker build -t runtest:0.0 .
Sending build context to Docker daemon  867.1MB
Step 1/3 : FROM ubuntu:16.04
 ---> 13c9f1285025
Step 2/3 : RUN mkdir /mydata
 ---> Running in 14e0adf98b47
Removing intermediate container 14e0adf98b47
 ---> 7989a8a7c405
Step 3/3 : RUN echo "Hello, Docker!"
 ---> Running in 7d40f263e424
Hello, Docker!
Removing intermediate container 7d40f263e424
 ---> 318b4bc1c3fd
Successfully built 318b4bc1c3fd
Successfully tagged runtest:0.0
[admin@centos7 ~]$
```

Docker 이미지 생성

- ***Dockerfile 작성***

  $ docker run –it –name runtest runtest:0.0

```
[admin@centos7 ~]$ docker run -it --name runtest runtest:0.0
root@081d5b715e4d:/# ls -al
total 4
drwxr-xr-x.   1 root root    6 Jun 20 03:05 .
drwxr-xr-x.   1 root root    6 Jun 20 03:05 ..
-rwxr-xr-x.   1 root root    0 Jun 20 03:05 .dockerenv
drwxr-xr-x.   2 root root 4096 Jun 10 20:41 bin
drwxr-xr-x.   2 root root    6 Apr 12  2016 boot
drwxr-xr-x.   5 root root  360 Jun 20 03:05 dev
drwxr-xr-x.   1 root root   66 Jun 20 03:05 etc
drwxr-xr-x.   2 root root    6 Apr 12  2016 home
drwxr-xr-x.   8 root root   96 Sep 13  2015 lib
drwxr-xr-x.   2 root root   34 Jun 10 20:41 lib64
drwxr-xr-x.   2 root root    6 Jun 10 20:40 media
drwxr-xr-x.   2 root root    6 Jun 10 20:40 mnt
drwxr-xr-x.   2 root root    6 Jun 20 03:04 mydata
drwxr-xr-x.   2 root root    6 Jun 10 20:40 opt
```

Docker 이미지 생성

- *Dockerfile 작성*

  6) *ADD command 추가*

     - *테스트용 파일 생성 (ex. test.txt)*

열기(O)

```
1 FROM ubuntu:16.04
2 ADD test.txt /
3
4
```

  7) *빌드*

     $ docker build –t addtest:0.0 .

```
[admin@centos7 ~]$ docker build -t addtest:0.0 .
Sending build context to Docker daemon  867.1MB
Step 1/2 : FROM ubuntu:16.04
 ---> 13c9f1285025
Step 2/2 : ADD test.txt /
 ---> 694fa902ec6a
Successfully built 694fa902ec6a
Successfully tagged addtest:0.0
[admin@centos7 ~]$
```

```
[admin@centos7 ~]$ docker run -it --name addtest addtest:0.0
root@f62e24ad3703:/# ls
bin   dev  home  lib64  mnt  proc  run   srv  test.txt  usr
boot  etc  lib   media  opt  root  sbin  sys  tmp       var
root@f62e24ad3703:/#
```

Docker 이미지 생성

njone company

- *Dockerfile 작성*

  8) ENTRYPOINT command 추가

  - *테스트용 파일 생성 (ex. test.sh)*

열기(O)

```
1 FROM ubuntu:16.04
2 ADD test.sh /
3 RUN chmod +x /test.sh
4 ENTRYPOINT /test.sh
5
```

7) 빌드

$ docker build –t entrytest:0.0 .

```
[admin@centos7 ~]$ docker build -t entrytest:0.0 .
Sending build context to Docker daemon  867.1MB
Step 1/4 : FROM ubuntu:16.04
 ---> 13c9f1285025
Step 2/4 : ADD test.sh /
 ---> Using cache
 ---> 77c89d0d72b5
Step 3/4 : RUN chmod +x /test.sh
 ---> Using cache
 ---> 961cb4f5c077
Step 4/4 : ENTRYPOINT /test.sh
 ---> Running in 6439e3c0b018
Removing intermediate container 6439e3c0b018
 ---> 6006685c3c3b
Successfully built 6006685c3c3b
Successfully tagged entrytest:0.0
[admin@centos7 ~]$
```

Docker 이미지 생성

- *Dockerfile 작성*

  8) 실행

  $ docker run -it --name entrytest entrytest:0.0

```
[admin@centos7 ~]$ docker run -it --name entrytest entrytest:0.0
Hello, Docker
[admin@centos7 ~]$
```

Docker 이미지 생성

- ***Lab) 다음 Node.js를 실행하는 Dockerfile 작성***

```
{
  "dependencies": {
    "express": "*"
  },
  "scripts": {
    "start": "node index.js"
  }
}
```

package.json

```
const express = require('express');

const app = express();

app.get('/', (req, res) => {
  res.send('How are you doing');
});

app.listen(8080, () => {
  console.log('Listening on port 8080');
});
```

index.js

$ npm install

$ npm start

Docker 이미지 생성

- ***Step1)***

```
1  FROM alpine
2
3  RUN npm install
4
5  CMD ["npm", "start"]
```

- ***Step2)***

```
1  FROM node:apline
2
3  RUN npm install
4
5  CMD ["npm", "start"]
```

- ***Step3)***

```
1  FROM node:apline
2
3  COPY ./package.json ./package.json
4  COPY ./index.js ./index.js
5
6  RUN npm install
7
8  CMD ["npm", "start"]
```

- ***Step4)***

```
1  FROM node:apline
2
3  WORKDIR /home/node
4
5  COPY ./package.json ./package.json
6  COPY ./index.js ./index.js
7
8  RUN npm install
9
10 CMD ["npm", "start"]
```

Docker 이미지 생성

열기(O)

```
1 FROM luiscortes/tomcat8.5-ora8jdk-alpineC
2
3 ENV TZ=Asia/Seoul
4 RUN ln -snf /usr/share/zoneinfo/$TZ /etc/localtime
5 RUN echo $TZ > /etc/timezone
6 RUN rm -Rf /usr/local/tomcat/webapps/ROOT
7 COPY target/post-1.0.0.war /usr/local/tomcat/webapps/ROOT.war
8 ENV JAVA_OPTS="-Dserver.type=dev"
```

$ docker build -t simple-app:1.0 .

$ docker run -it --name simple-app-1 simple-app:1.0

$ docker exec -it simple-app-1 /bin/bash

→ ip address 확인

Lab – Docker 이미지 생성

- *Dockerfile 실습*
  - *python 이미지 배포*
    - *Base Image → python:3.7.9-stretch 사용*

    job1) numpy를 설치하는 명령어를 추가하시오.

    - pip install –y numpy

    job2) numpy 예제를 실행하기 위해 test.py 파일을 생성하시오.

    job3) test.py 파일을 이미지 내로 복사하고 실행하는 명령어를 추가하시오.

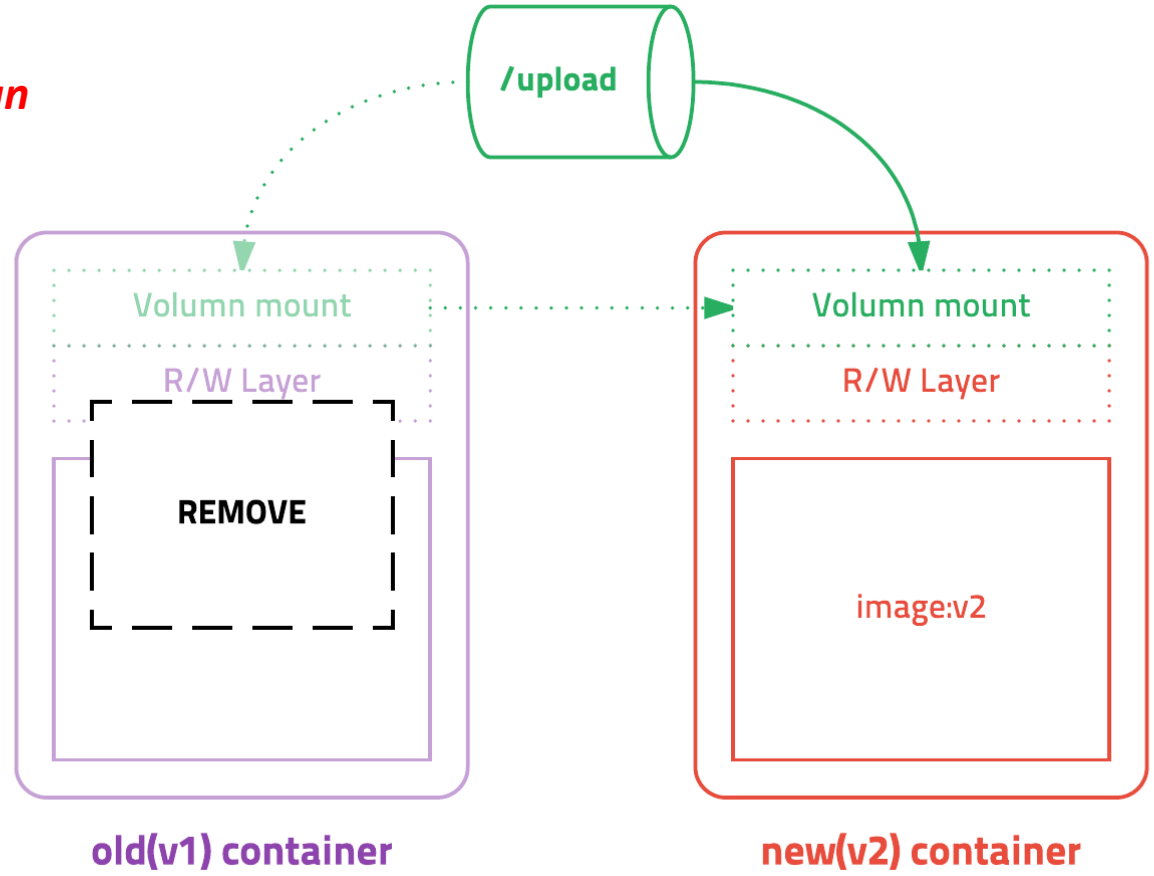
    - python test.py

Docker 공유 폴더 (volume)

- 컨테이너 업데이트
  - 새 버전의 이미지 다운 → ***pull***
  - 기존 컨테이너 삭제 → ***stop, rm***
  - 새 이미지를 이용하여 새 컨테이너 실행 → ***run***

- 컨테이너 유지 정보
  - ***AWS S3***
  - ***데이터 볼륨***

```
$ docker run –d -p 3306:3306 \
  -e MYSQL_ALLOW_EMPTY_PASSWORD=true \
  --name mysql \
  -v /my/datadir:/var/lib/mysql
  mysql:5.7
```



Lab – MongoDB 설치

1) 이미지 받아오기

$ docker pull mongo (or $ docker pull mongo:4.1)

2) 기동

```
$ docker run –name mongodb_server -v /home/test/mongodb/db:/data/db \
      -d -p 16010:27017 mongo –auth
$ docker run -d --name mongodb-test -p 17017:27017 \
      -v /home/sa/data/mongod.conf:/etc/mongod.conf \
      -v /home/sa/data/db:/data/db mongo --config /etc/mongod.conf
```

3) Bash 접근, Mongo 접속

$ docker exec -it mongodb_server bash

root@XXXXXXXXX $ mongo

Lab – MongoDB 설치

4) 관리자 계정 생성

mongo> use admin

5) 관리자 로그인, 일반 계정 생성

mongo> db.createUser({

user: ”admin”, pwd: “admin”, roles: [{ role: “userAdminAnyDatabase”, db: “admin” }]

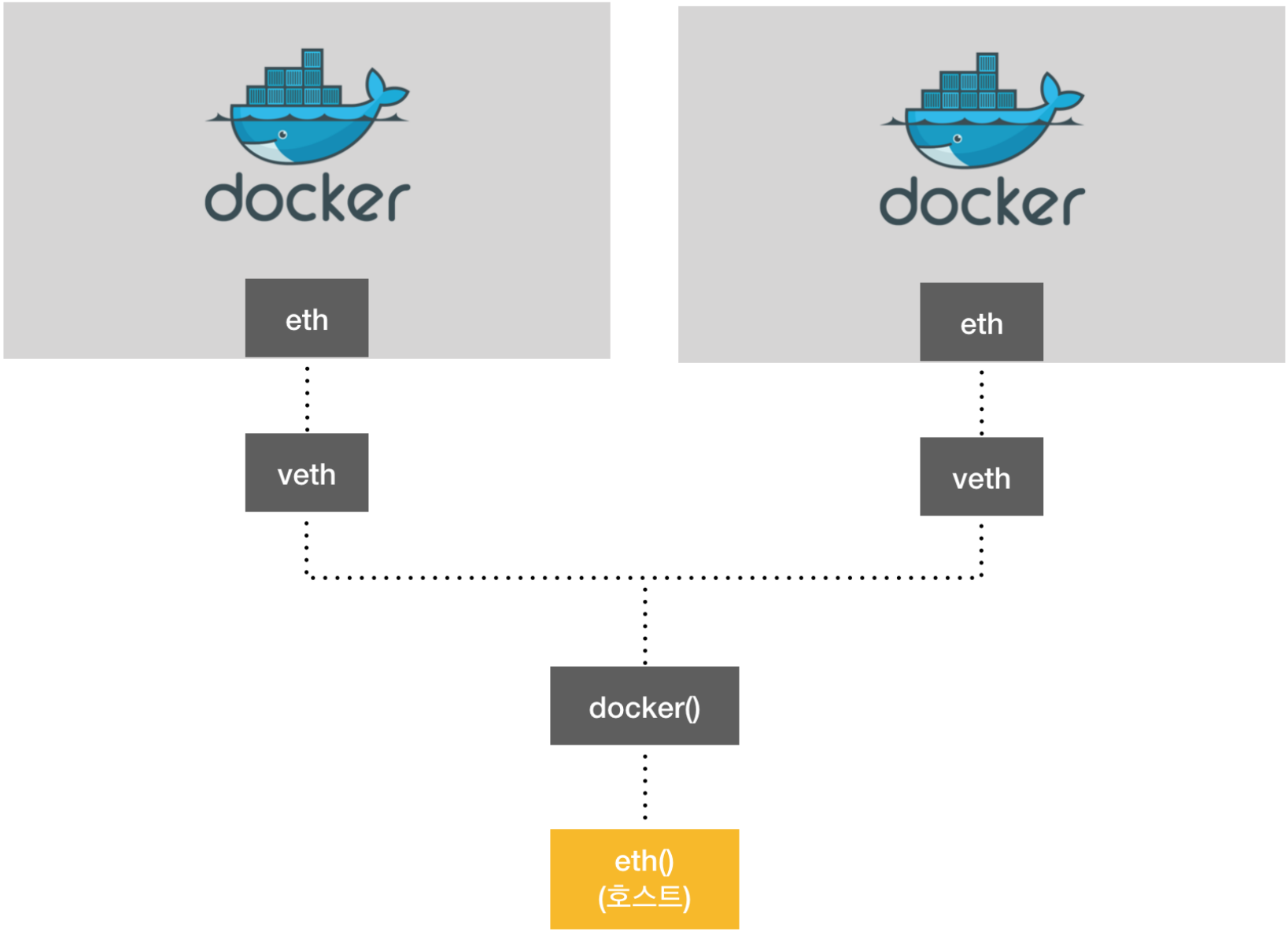
})

6) ReplicaSet 설정

$ mongo -u “admin” -p “admin” –authenticationDatabase “admin”

mongo> use testdb db.createUser({ user: “tester”, pwd: “1234”, roles: [“dbAdmin”, “readWrite”] })

Docker 네트워크



Docker 네트워크

- ***Bridge network***
  - $ docker network create --driver bridge [브릿지 이름]
  - $ docker run 또는 docker create에서 --net 옵션으로 커스텀 브릿지 사용
- ***Host network***
  - 네트워크를 호스트로 설정하면 호스트의 네트워크 환경을 그대로 사용
  - 포트 포워딩 없이 내부 어플리케이션 사용
  - $ docker run -it --name network_host --net host ubuntu:16.04

Docker 네트워크

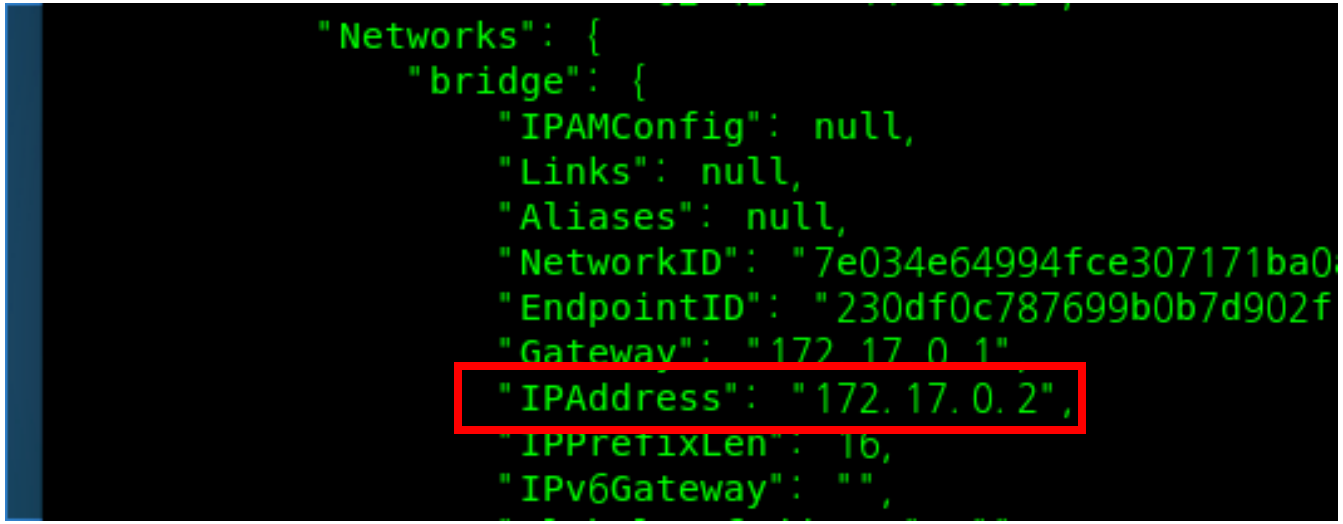
- ***None network***
  - 네트워크를 사용하지 않음
  - lo 네트워크만 사용, 외부와 단절
- ***Container network***
  - 다른 컨테이너의 네트워크 환경 공유
  - --net container:[다른 컨테이너의 ID or Name]
  - 두 컨테이너 사이의 네트워크 정보가 동일
- ***Overlay network***
  - 다른 호스트들 간에 네트워크 공유

Docker 네트워크

- *docker network ls*
- *docker network inspect [netword-id]*
- *docker network create my-network*
  - *docker network rm my-network*
- *docker network connect my-network (컨테이너)*
- *docker network disconnect my-network (컨테이너)*

Docker Inspect

- *Tomcat → Mysql Connection*
  - *$ docker inspect <tomcat image>*
  - *$ docker inspect <mysql image>*

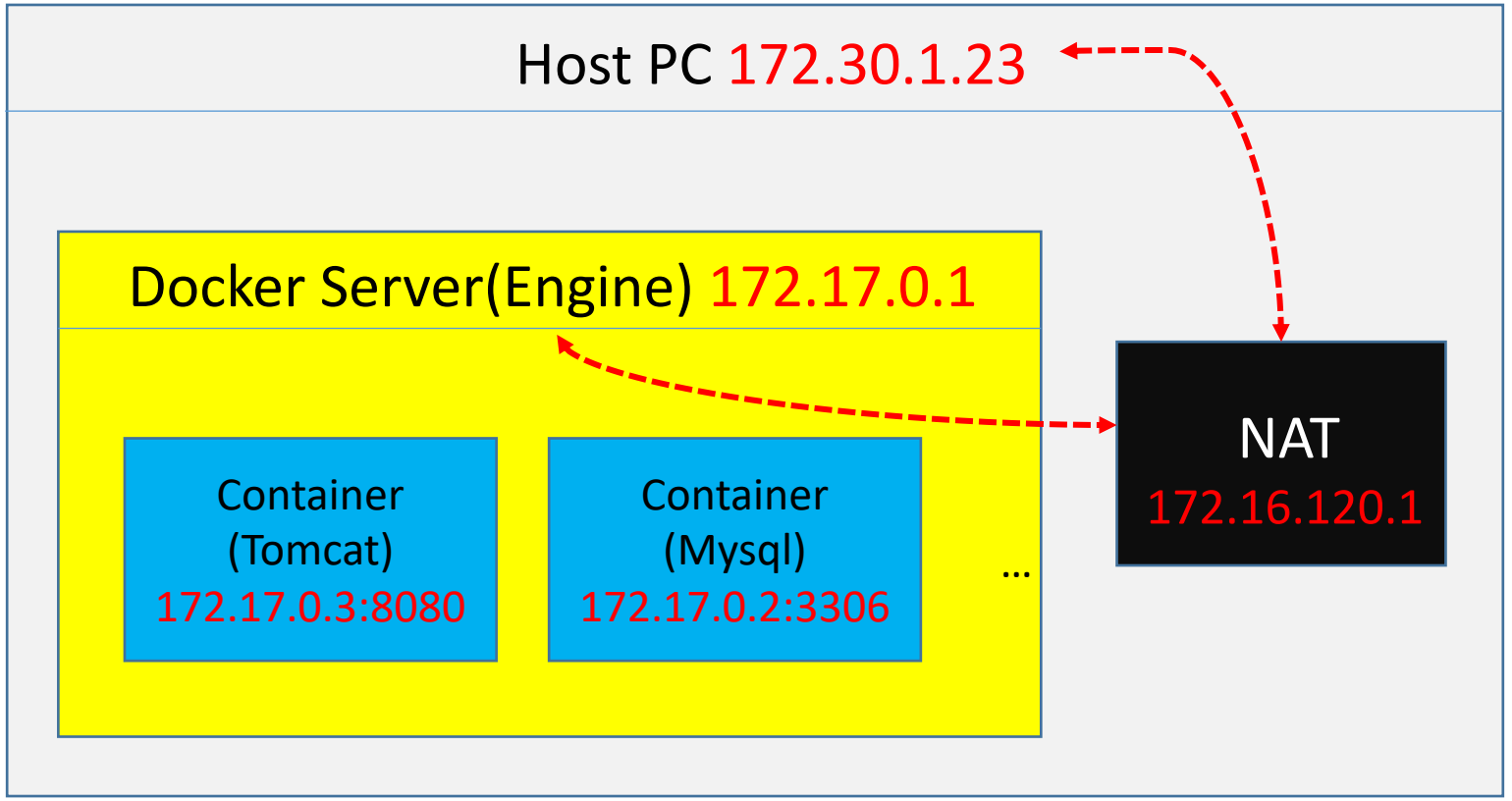


- *application.properties 파일*

```
spring.datasource.url=jdbc:mysql://172.17.0.2:3306/testdb?serverTimezone=UTC
```

Docker Inspect

- ***Tomcat → Mysql Connection***



Lab – Docker 네트워크

- ***아래와 같은 구성을 위한 Docker Network 를 설계하시오.***

  1) Tomcat + Mysql 연동 테스트
  2) Python + Mysql 연동 테스트
  3) Node.js + Python(or Tomcat) 연동 테스트

Docker Registry

- Private registry를 설치하기 위해 사용되는 Docker Official Iamge
- Docker registry 설치

  $ docker run -d -p 5000:5000 --restart always --name registry registry:2

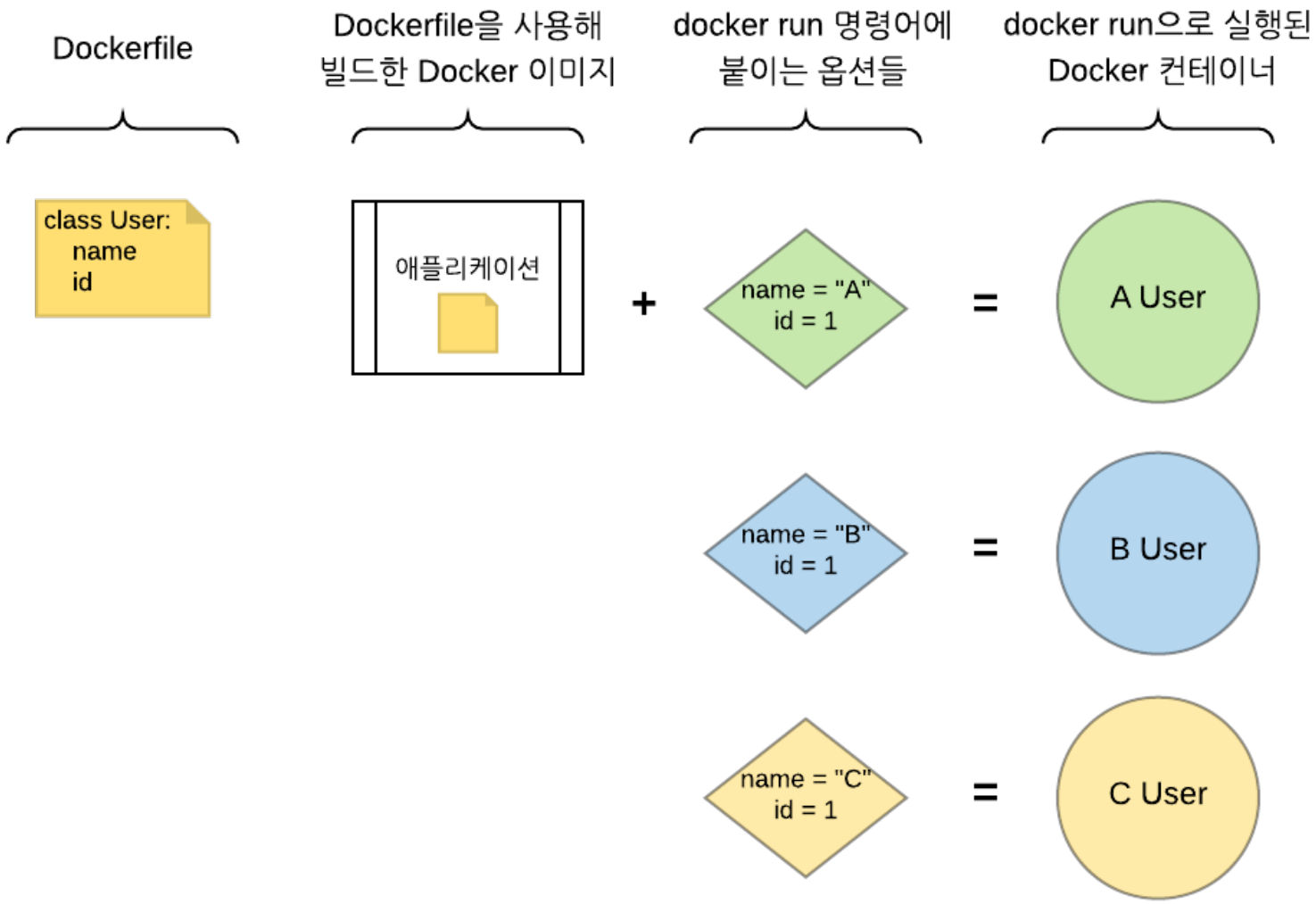
  $ curl –X GET http://localhost:5000/v2/_catalog

- Private registry를 위한 Docker build

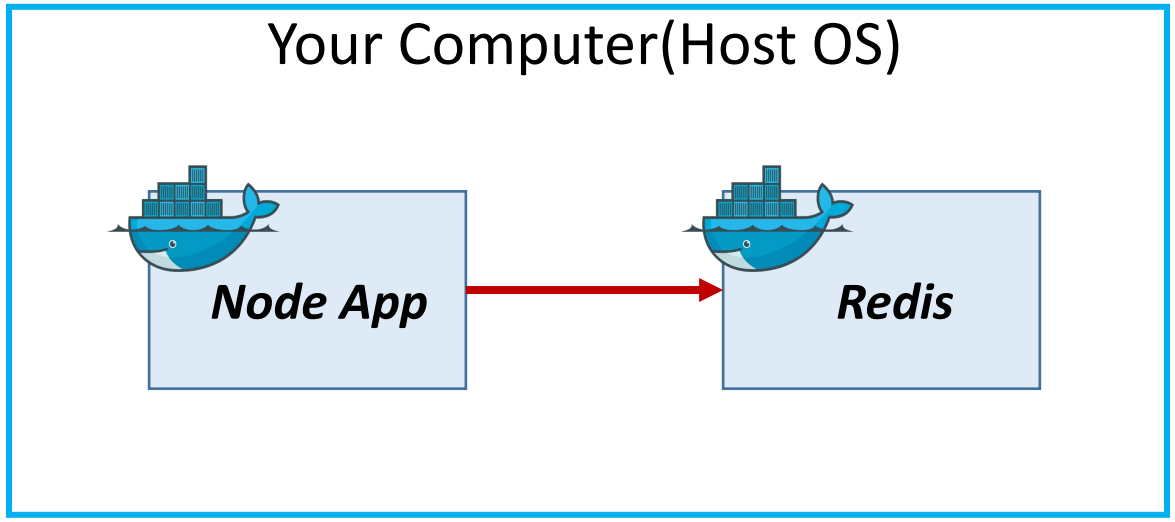
  $ docker build --tag http://localhost:5000/***[이미지명]:[TAG 명]*** -f Dockerfile .

Docker Compose

- 기존의 Dockerfile을 통한 Container 실행 방법



Docker Compose

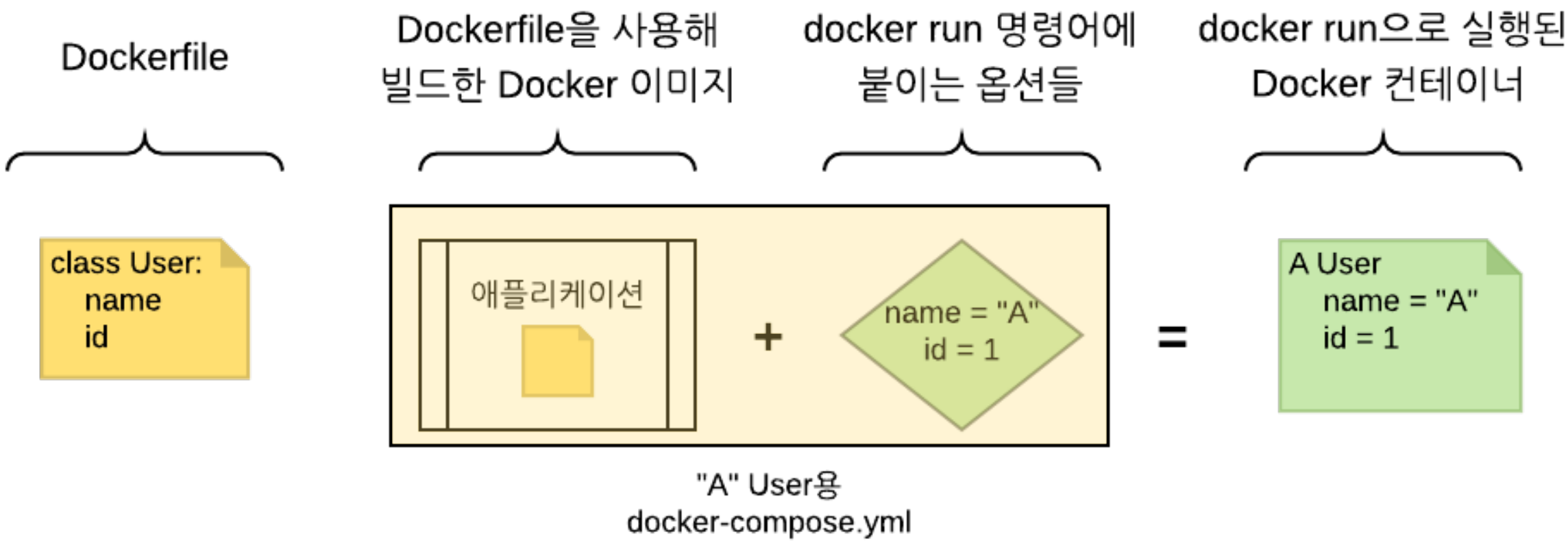


- ***다른 컨테이너 접속***
  - Docker CLI's Networking Features
  - *Use Docker Compose*

- 도커 애플리케이션을 정의하고 실행하는 도구
- Tool for defining and running ***multi-container*** **Docker applications**)
  - 각 컨테이너별로 별도의 Docker 명령어 실행 (long-winded arguments)
  - 한 번에 여러 개의 컨테이너 동시 실행

Docker Compose

- ***Docker 커맨드*** or 복잡한 ***설정***을 쉽게 관리하기 위한 ***도구***
  - ***YAML*** format에 Docker 생성 , 설정 관련 된 작업을 작성해 놓은 Script 파일



Docker Compose

- ***Docker 커맨드*** or 복잡한 ***설정***을 쉽게 관리하기 위한 ***도구***
  - ***YAML*** format에 Docker 생성 , 설정 관련 된 작업을 작성해 놓은 Script 파일

```
version: '3.1'

service:
        servicename:
                image: # optional
                command: # optional
                environment: # optional
                volumes # optional
        servicename2: # if have second service...
volumes: # optional

network: # optional
```

- ***Docker compose 실행***

  $ docker-compose up

  $ docker-compose up --build → Dockerfile을 다시 빌드

Docker Compose

- ***Docker compose를 이용한 MongoDB 설치***

```
-rw-rw-r-- 1 ubuntu ubuntu  184 Apr 16 13:38 docker-compose.yml
drwxrwxr-x 2 ubuntu ubuntu 4.0K Apr 16 13:30 mongo1
drwxrwxr-x 2 ubuntu ubuntu 4.0K Apr 16 13:30 mongo2
drwxrwxr-x 2 ubuntu ubuntu 4.0K Apr 16 13:30 mongo3
drwxrwxr-x 2 ubuntu ubuntu 4.0K Apr 16 13:31 setup
```

```
config = {
    _id : "replication",
    members: [
        {_id:0,host : "mongo1:27017"},
        {_id:1,host : "mongo2:27017"},
        {_id:2,host : "mongo3:27017"},
    ]
}

rs.initiate(config);

rs.conf();
```

replSet.js

Docker Compose

- ***Docker compose를 이용한 MongoDB 설치***
  - 해당 서비스가 실행되고 10초 뒤에 mongo 접속하여 replicaSet 를 해주기 위한 쉘스크립트

```
#!/bin/bash
sleep 10 | echo Sleeping
mongo mongodb://mongo1:27017 replicaSet.js
```

setup.sh

```
FROM mongo

WORKDIR /usr/src
RUN mkdir configs
WORKDIR /usr/src/configs

COPY replicaSet.js .
COPY setup.sh .

RUN chmod +x ./setup.sh

CMD ["./setup.sh"]
```

Dockerfile

Docker Compose

njone company

- ***Docker compose를 이용한 MongoDB 설치***
  - 이미지 빌드

  $ docker build -t ***mongo_repl_setup*** .

```
Sending build context to Docker daemon  4.096kB
Step 1/8 : FROM mongo
 ---> 9a63ed32fc2b
Step 2/8 : WORKDIR /usr/src
 ---> Using cache
 ---> cc30aeaa2b1b
Step 3/8 : RUN mkdir configs
 ---> Using cache
 ---> 3ff2c6dc0d01
Step 4/8 : WORKDIR /usr/src/configs
 ---> Using cache
 ---> cb51fdaa3b99
Step 5/8 : COPY replicaSet.js .
 ---> 5826a1021077
Step 6/8 : COPY setup.sh .
 ---> beb2ecdedc2a
Step 7/8 : RUN chmod +x ./setup.sh
 ---> Running in a25c2c893198
Removing intermediate container a25c2c893198
 ---> ada73d6fe757
Step 8/8 : CMD ["./setup.sh"]
 ---> Running in 8109bf3a3d00
Removing intermediate container 8109bf3a3d00
 ---> 7cf5567c627e
Successfully built 7cf5567c627e
Successfully tagged setup-rspl:latest
```

Docker Compose

- ***Docker compose를 이용한 MongoDB 설치***

  - Docker compose 실행 → docker-compose.yml 준비

  $ docker-compose up

  ~

  $ docker ps

  $ docker exec -it <Container ID> mongo

  → 3대 중 1대는 Primary, 2대는 Secondary

- ***확인***

  1) 테스트 데이터베이스 생성, 데이터 추가
  2) Primary 중지 → docker stop <Container ID>
  3) 다른 Secondary 가 Primary로 승격

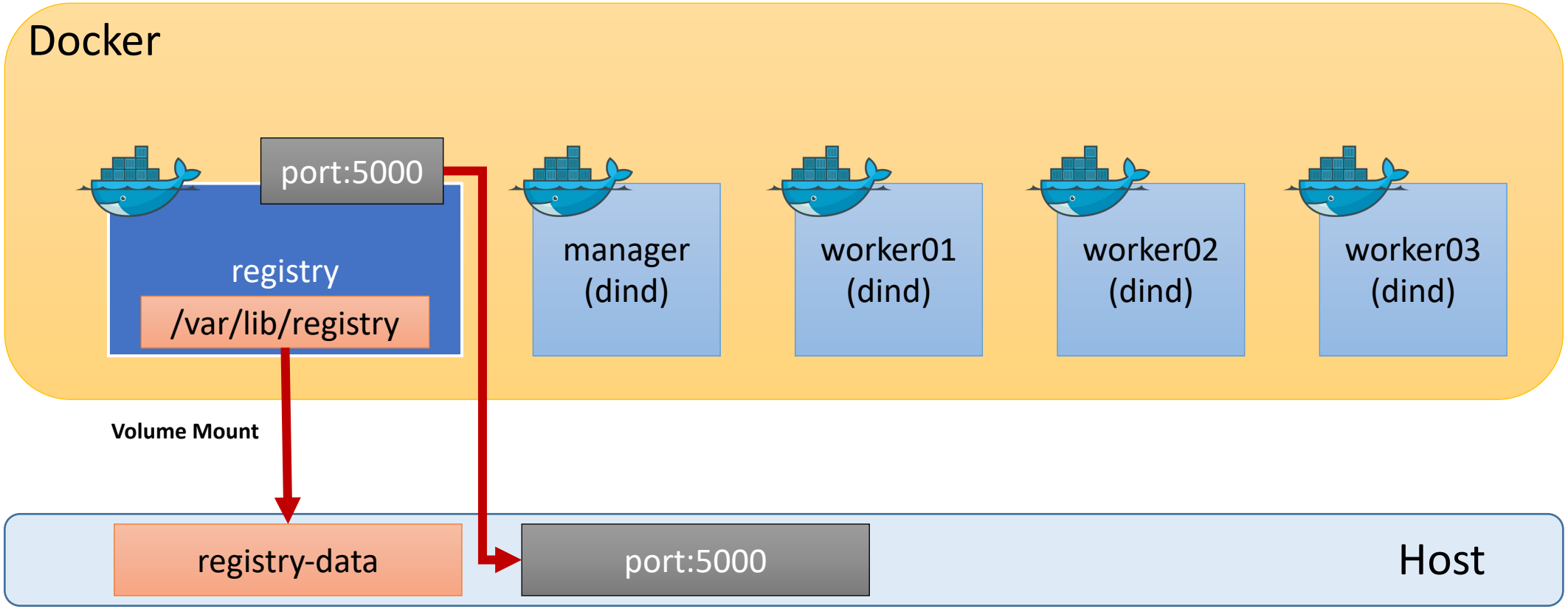
Docker Swarm

- ***Kubernetes 이전에 사용되었던 Orchestration tool***
- ***Docker Swarm → 오케스트레이션***
  - ***여러 docker host를 클러스터로 묶어주는 컨테이너 오케스트레이션***

이름	역할	대응하는 명령어
Compose	여러 컨테이너로 구성된 도커 애플리케이션을 관리 (주로 단일 호스트)	docker-compose
Swarm	클러스터 구축 및 관리 (주로 멀티 호스트)	docker swarm
Service	스웜에서 클러스트 안의 서비스 (컨테이너 하나 이상의 집합)을 관리	docker service
Stack	스웜에서 여러 개의 서비스를 합한 전체 애플리케이션을 관리	docker stack

Docker Swarm

- ***Docker in Docker, dind***
  - *도커 컨테이너 안에서 도커 호스트를 실행*



Docker Swarm

- ***Manager에 Swarm 설정***
  - Manager container에 docker swarm init 설정 → Swarm 모드 활성화
  - Worker 컨테이너를 등록 ***with join token***

$ docker exec -it manager ***docker swarm init***

$ docker exec -it ***work01 docker swarm join*** \

--token ***[JOIN TOKEN ex) SWMTKN-1-4u47j~bo0]*** manager:2377

→ worker02, worker03 도 등록

- Swarm에서 사용할 포트
  - TCP port 2377: cluster management 통신에 사용
  - TCP/UDP port 7946: node간의 통신에 사용
  - TCP/UDP port 4789: overlay network 트래픽에 사용

Docker Swarm

- ***도커 레지스트리용 이미지 생성***

$ docker tag example/echo:latest ***localhost:5000/example/echo:latest***

[registry_host/]***repository_name***[:tag]

- ***도커 레지스트리에 이미지 등록***

$ docker push ***localhost:5000/example/echo:latest***

- ***Worker 컨테이너에 이미지 설치***

$ docker exec –it worker01 ***docker pull registry:5000/example/echo:latest***

$ docker exec –it worker01 ***docker image ls***

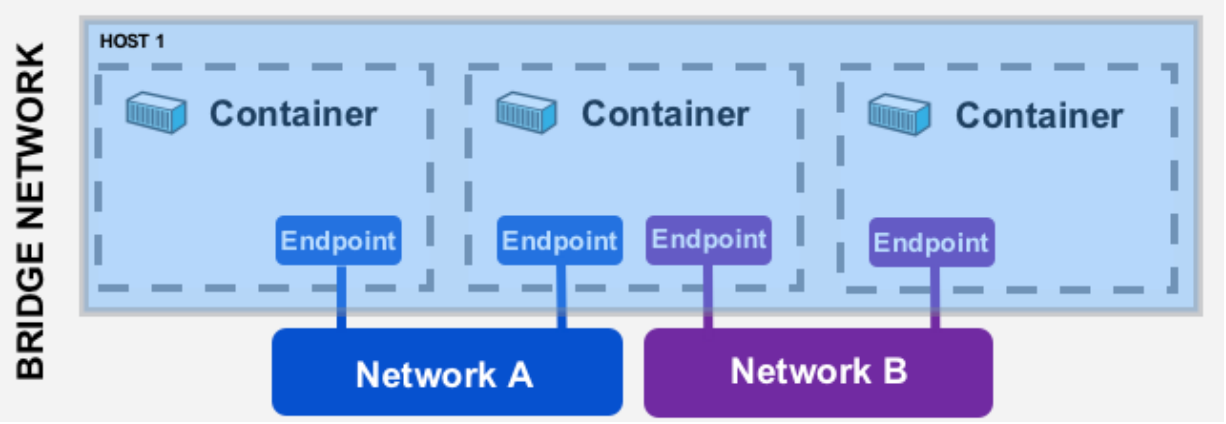
Docker Service

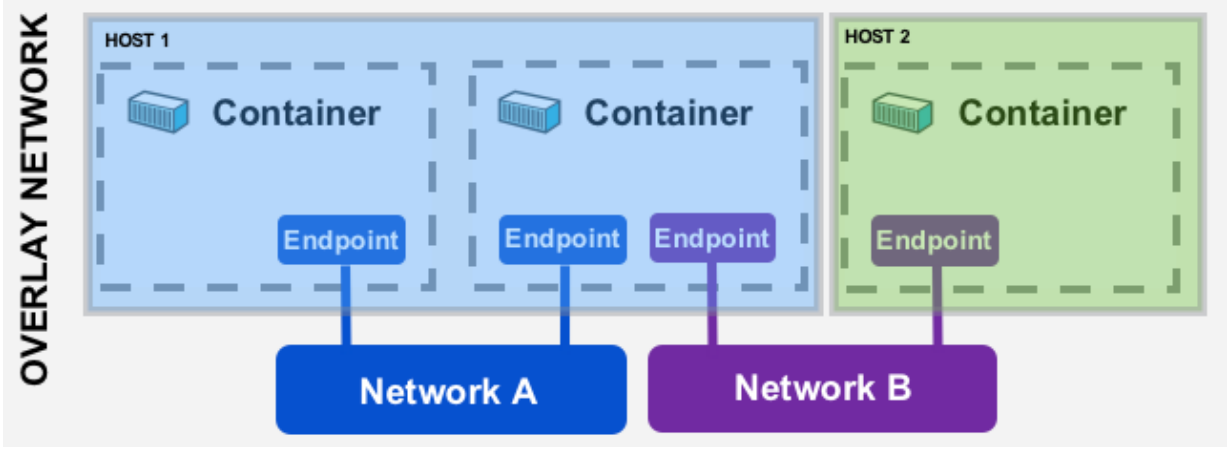
- ***Service***
  - 애플리케이션을 구성하는 일부 컨테이너(단일 또는 복수)를 제어하기 위한 단위

```
$ docker exec -it manager \
        docker service create --replicas 1 --publish 8000:8080 --name echo\
        registry:5000/example/echo:latest
$ docker exec -it manager docker service ls
$ docker exec -it manager docker service scale echo=6
$ docker exec -it manager docker service ps echo
$ docker exec -it manager docker service rm echo
```

Docker Stack

- ***Stack***
  - 하나 이상의 서비스를 그룹으로 묶은 단위, 애플리케이션 전체 구성 정의
    - 서비스는 애플리케이션 이미지를 하나 밖에 다루지 못함
  - 여러 서비스를 함께 다룰 수 있음
  - 스택을 사용해 배포된 서비스 그룹은 ***overlay 네트워크***에 속함





Docker Stack

- ***Overlay 네트워크 생성***

$ docker exec -it manager ***docker network create*** --driver=overlay --attachable ***ch03***

```yaml
version: "3"
services:
  nginx:
    image: gihyodocker/nginx-proxy:latest
    deploy:
      replicas: 3
      placement:
        constraints: [node.role != manager]
    environment:

      BACKEND_HOST: echo_api:8080
    depends_on:
      - api
    networks:
      - ch03
```

```yaml
  api:
    image: registry:5000/example/echo:latest
    deploy:
      replicas: 3
      placement:
        constraints: [node.role != manager]
    networks:
      - ch03

networks:
  ch03:
    external: true
```

Docker Stack

- ***Stack 배포***

  - volume으로 설정 된 /stack/ 폴더의 ch03-webapi.yml 파일 실행

  $ docker exec -it manager ***docker stack deploy*** -c /stack/ch03-webapi.yml ***echo***

- ***배포된 Stack 확인***

  $ docker exec -it manager ***docker stack services echo***

- ***Stack에 배포된 컨테이너 확인***

  $ docker exec -it manager ***docker stack ps echo***

- ***Stack 삭제***

  $ docker exec -it manager ***docker stack rm echo***

Docker Stack

- ***Visualizer를 사용해 컨테이너 배치 시각화***
  - ***dockersamples/visualizer 이미지로 배포 됨***

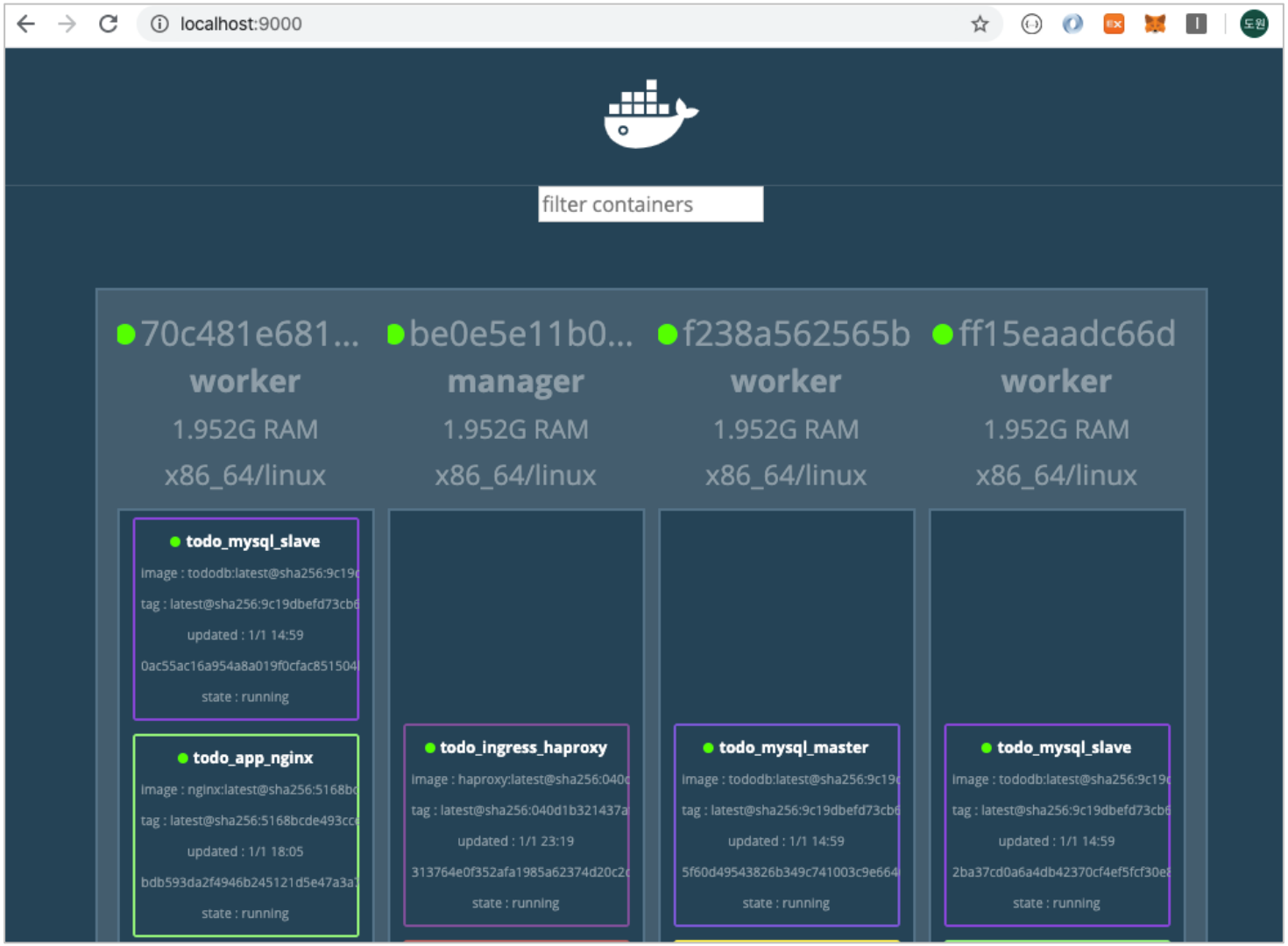
```yaml
version: "3"

services:
  app:
    image: dockersamples/visualizer
    ports:
      - "9000:8080"
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    deploy:
      mode: global
      placement:
        constraints: [node.role == manager]
```

- ***Visualizer stack 배포***

$ docker exec -it manager ***docker stack deploy*** -c /stack/visualizer.yml visualizer

Docker Stack

- ***Visualizer를 사용해 컨테이너 배치 시각화***



Docker HAProxy

- ***Visualizer는 외부 호스트에서 접속 가능***
  - Host → Manager 사이는 ***port forwarding*** 설정

- ***HAProxy***
  - 외부 호스트에서 요청되는 트래픽을 목적 서비스로 보내주는 프록시 서버 설정
  - dockercloud/haproxy 이미지로 배포
    - 컨테이너 외부에서 서비스에 접근할 수 있도록 해 주는 다리 역할 (***ingress***)
    - 서비스가 배치된 노드에 로드 밸런싱 기능 제공

Docker HAProxy

- ***HAProxy***

```yaml
version: "3"

services:
  haproxy:
    image: dockercloud/haproxy
    networks:
      - ch03
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    deploy:
      mode: global
      placement:
        constraints:
          - node.role == manager
    ports:
      - 80:80
      - 1936:1936 # for stats page (basic auth. stats:stats)

networks:
  ch03:
    external: true
```

ch03-ingress.yml

```yaml
  environment:
    SERVICE_PORTS: 80
    BACKEND_HOST: echo_api:8080
```

ch03-webapi.yml 수정

Docker HAProxy

- ***HAProxy***

$ docker exec -it manager ***docker stack deploy*** -c /stack/ch03-webapi.yml ***echo***

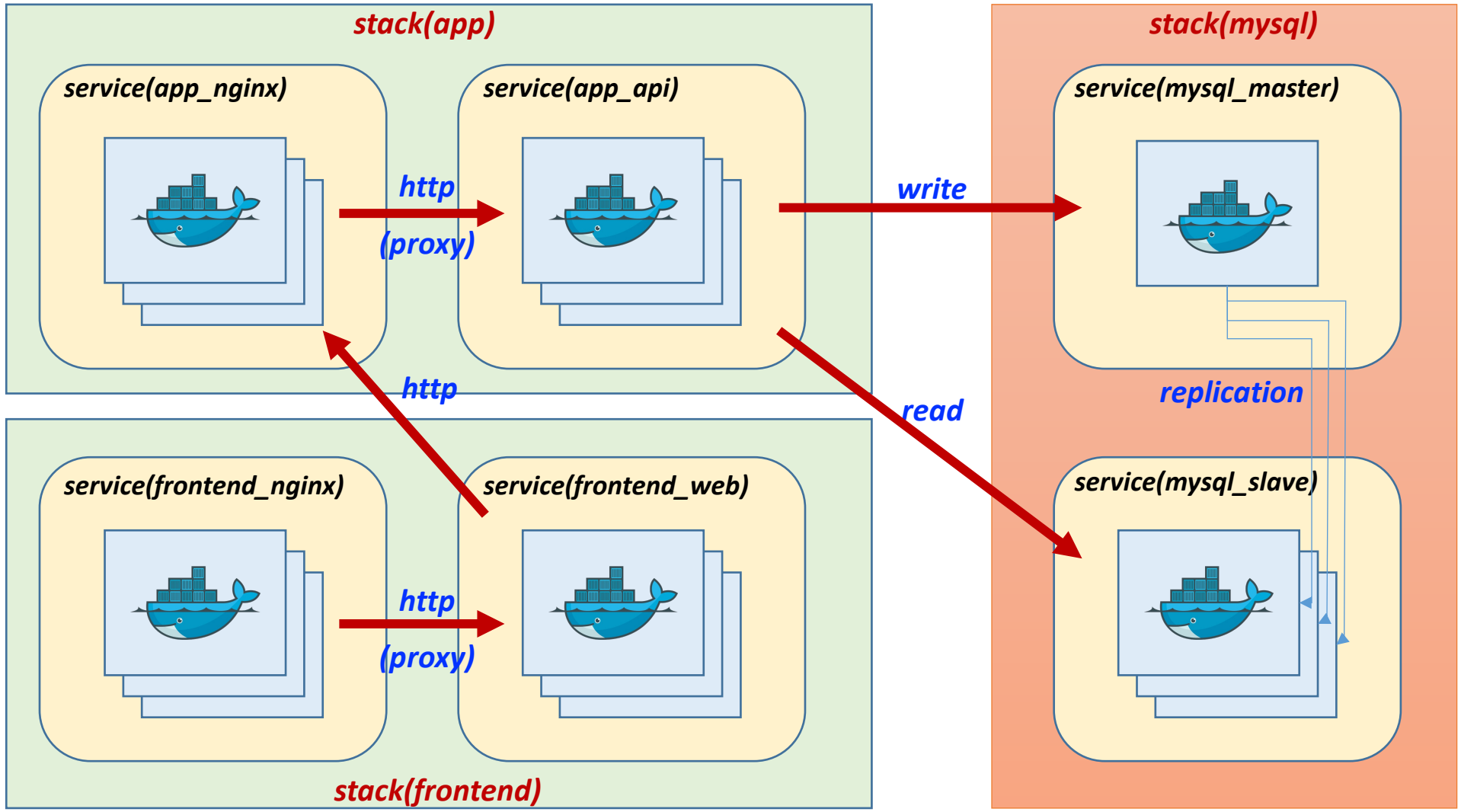
$ docker exec -it manager ***docker stack deploy*** -c /stack/ch03-ingress.yml ***ingress***

$ docker exec -it manager ***docker service ls***

```
$ curl http://localhost:8000
```

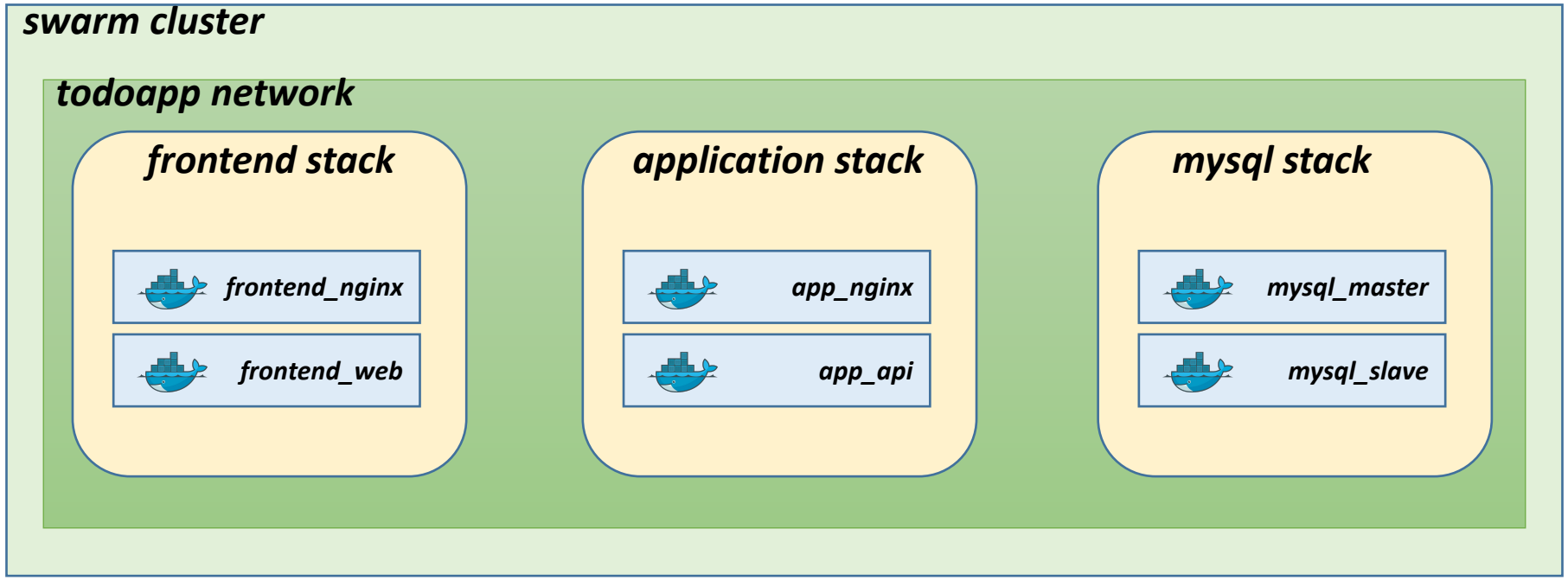
Swarm을 이용한 실전 애플리케이션 개발

- *아키텍처*



Swarm을 이용한 실전 애플리케이션 개발

- ***배치전략***



- 데이터 스토어 역할을 할 MySQL 서비스를 Master-Slave 구조로 구축
- MySQL과 데이터를 주고 받을 API 구현
- Nginx를 웹 애플리케이션과 API 사이에서 리버스 프록시 역할을 하도록 설정
- API를 사용해 서버 사이드 랜더링을 수행할 웹 애플리케이션 구현
- 프론트앤드 쪽에 리버스 프록시(Nginx) 배치

- ***Master/Slave 이미지 생성***
- ***컨테이너의 설정 파일 및 스크립트 다루는 방법***
- ***데이터베이스 초기화***
- ***Master – Slave 간의 Replication 설정***

1) Master/Slave 구조 구축

- Docker hub의 Mysql:5.7 이미지로 생성
- Master/Slave 컨테이너는 두 역할을 모두 수행할 수 있는 하나의 이미지로 생성
- MYSQL_MASTER 환경 변수의 유무에 따라 Master, Slave 결정
- replicas 값을 설정하여 Slave 개수 조정

- *인증 정보*

Master

환경 변수 이름	내용	값
MYSQL_ROOT_PASSWORD	root 사용자 패스워드	gihyo
MYSQL_DATABASE	TODO 애플리케이션에서 사용할 데이터베이스	tododb
MYSQL_USER	데이터베이스 사용자명	gihyo
MYSQL_PASSWORD	패스워드	gihyo

Slave

환경 변수 이름	내용	값
MYSQL_MASTER_HOST	Master 호스트명	master
MYSQL_ROOT_PASSWORD	root 사용자 패스워드	gihyo
MYSQL_DATABASE	TODO 애플리케이션에서 사용할 데이터베이스	tododb
MYSQL_USER	데이터베이스 사용자명	gihyo
MYSQL_PASSWORD	패스워드	gihyo
MYSQL_REPL_USER	Master에 등록할 Replication 사용자	repl
MYSQL_REPL_PASSWORD	레플리케이션 사용자 패스워드	gihyo

- *MySQL 설정 파일*

```
1  [mysqld]
2  character-set-server = utf8mb4
3  collation-server = utf8mb4_general_ci
4  pid-file    = /var/run/mysqld/mysqld.pid
5  socket      = /var/run/mysqld/mysqld.sock
6  datadir     = /var/lib/mysql
7  #log-error  = /var/log/mysql/error.log
8  # By default we only accept connections from localhost
9  #bind-address   = 127.0.0.1
10 # Disabling symbolic-links is recommended to prevent assorted security risks
11 symbolic-links=0
12 relay-log=mysqld-relay-bin
13 relay-log-index=mysqld-relay-bin
14
15 log-bin=/var/log/mysql/mysql-bin.log
16
17
```

→ *log-bin*

→ *server-id*

etc/mysql/mysql.conf.d/mysqld.conf

```
1  #!/bin/bash -e
2  OCTETS=(`hostname -i | tr -s '.' ' '`)
3
4  MYSQL_SERVER_ID=`expr ${OCTETS[2]} \* 256 + ${OCTETS[3]}`
5  echo "server-id=$MYSQL_SERVER_ID" >> /etc/mysql/mysql.conf.d/mysqld.cnf
```

add-server-id.sh

- ***Replication 설정***

```
# (1) 환경 변수로 마스터와 슬레이브를 지정
if [ ! -z "$MYSQL_MASTER" ]; then
  echo "this container is master"
  return 0
fi

echo "prepare as slave"

# (2) 슬레이브에서 마스터와 통신 가능 여부 확인
if [ -z "$MYSQL_MASTER_HOST" ]; then
  echo "mysql_master_host is not specified" 1>&2
  return 1
fi
```

```
# (3) 마스터에 레플리케이션용 사용자 생성 및 권한 부여
IP=`hostname -i`
IFS='.'
set -- $IP
SOURCE_IP="$1.$2.%.%"
mysql -h $MYSQL_MASTER_HOST -u root -p$MYSQL_ROOT_PASSWORD -e "CREATE USER IF NOT EXISTS '$MYSQL_REPL_USER'@'$SOURCE_IP'
IDENTIFIED BY '$MYSQL_REPL_PASSWORD';"
mysql -h $MYSQL_MASTER_HOST -u root -p$MYSQL_ROOT_PASSWORD -e "GRANT REPLICATION SLAVE ON *.* TO
'$MYSQL_REPL_USER'@'$SOURCE_IP';"
```

```
# (5) 레플리케이션 시작
mysql -u root -p$MYSQL_ROOT_PASSWORD -e "CHANGE MASTER TO MASTER_HOST='$MYSQL_MASTER_HOST', MASTER_USER='$MYSQL_REPL_USER',
MASTER_PASSWORD='$MYSQL_REPL_PASSWORD', MASTER_LOG_FILE='$BINLOG_FILE', MASTER_LOG_POS=$BINLOG_POSITION;"
mysql -u root -p$MYSQL_ROOT_PASSWORD -e "START SLAVE;"
```

prepare.sh

- ***Dockerfile – mysql_master, mysql_slave***

```
1  FROM mysql:5.7
2
3  # (1) 패키지 업데이트 및 wget 설치
4  RUN apt-get update
5  RUN apt-get install -y wget
6
7  # (2) entrykit 설치
8  RUN wget https://github.com/progrium/entrykit/releases/download/v0.4.0/entrykit_0.4.0_linux_x86_64.tgz
9  RUN tar -xvzf entrykit_0.4.0_linux_x86_64.tgz
10 RUN rm entrykit_0.4.0_linux_x86_64.tgz
11 RUN mv entrykit /usr/local/bin/
12 RUN entrykit --symlink
13
14 # (3) 스크립트 및 각종 설정 파일 복사
15 COPY add-server-id.sh /usr/local/bin/
16 COPY etc/mysql/mysql.conf.d/mysqld.cnf /etc/mysql/mysql.conf.d/
17 COPY etc/mysql/conf.d/mysql.cnf /etc/mysql/conf.d/
18 COPY prepare.sh /docker-entrypoint-initdb.d
19 COPY init-data.sh /usr/local/bin/
20 COPY sql /sql
21
22 # (4) 스크립트, mysqld 실행
23 ENTRYPOINT [ \
24   "prehook", \
25     "add-server-id.sh", \
26     "--", \
27   "docker-entrypoint.sh" \
28 ]
29
30 CMD ["mysqld"]
```

- *컨테이너 실행 시 처리할 내용을 기술*
- *주 프로세스보다 먼저 실행할 명령 처리*

필요 시 실행 권한 부여

- ***Build 및 Swarm cluster에서 사용하기***
  - Dockerfile 빌드 → ch04/tododb:latest 이미지 생성
  - 생성 된 이미지를 Swarm cluster의 Worker node에서 사용할 수 있도록 ***localhost:5000/ch04/tododb:latest*** 태그를 붙여 레지스트리에 등록

```
$ docker  build  -t  ch04/tododb:latest .
$ docker  tag  ch04/tododb:latest  localhost:5000/ch04/tododb:latest
$ docker  push localhost:5000/ch04/tododb:latest
```

- ***Registry의 이미지 목록 → http://localhost:5000/v2/_catalog***

Swarm을 이용한 실전 애플리케이션 개발 - MySQL 서비스 구축

$ docker exec -it manager docker ***network create*** --driver=***overlay*** --attachable todoapp

- ***Swarm 에서 Master/Slave 실행***
  - MySQL master x 1(replicas=1), MySQL slave x 2(replicas=2)

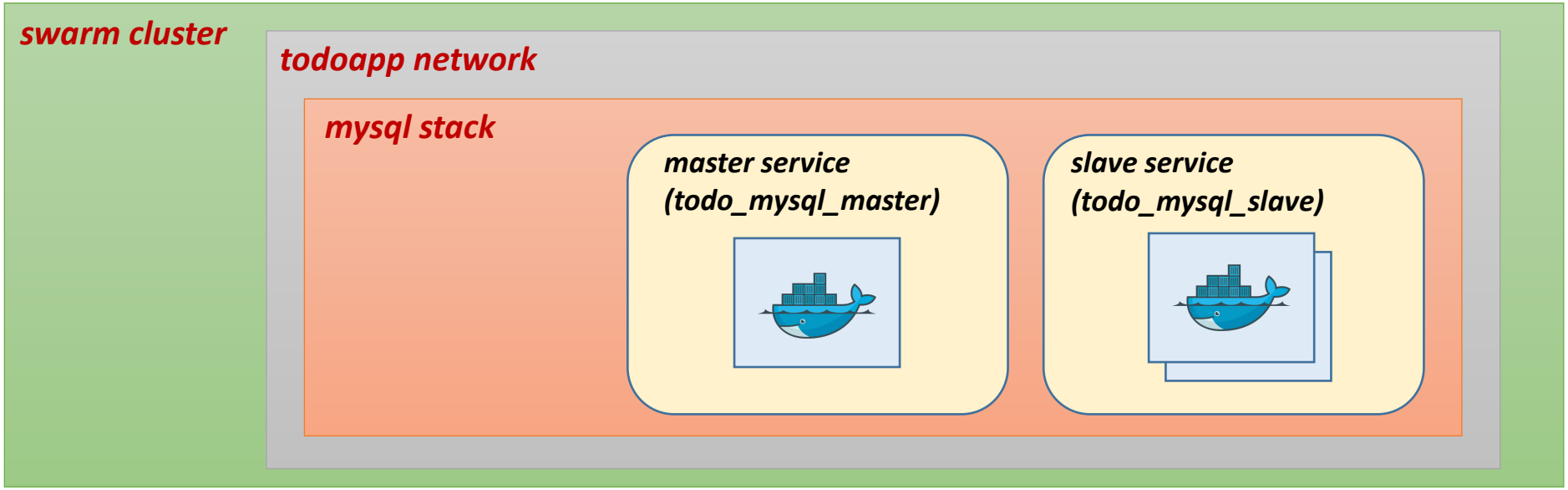
```yaml
version: "3"

services:
  master:
    image: registry:5000/ch04/tododb:latest
    deploy:
      replicas: 1
      placement:
        constraints: [node.role != manager]
    environment:
      MYSQL_ROOT_PASSWORD: gihyo
      MYSQL_DATABASE: tododb
      MYSQL_USER: gihyo
      MYSQL_PASSWORD: gihyo
      MYSQL_MASTER: "true"
    networks:
      - todoapp

  slave:
    image: registry:5000/ch04/tododb:latest
    deploy:
      replicas: 2
      placement:
        constraints: [node.role != manager]
    depends_on:
      - master
    environment:
      MYSQL_MASTER_HOST: master
      MYSQL_ROOT_PASSWORD: gihyo
      MYSQL_DATABASE: tododb
      MYSQL_USER: gihyo
      MYSQL_PASSWORD: gihyo
      MYSQL_ROOT_PASSWORD: gihyo
      MYSQL_REPL_USER: repl
      MYSQL_REPL_PASSWORD: gihyo
    networks:
      - todoapp

networks:
  todoapp:
    external: true
```

todo-mysql.yml

Swarm을 이용한 실전 애플리케이션 개발 - MySQL 서비스 구축

- ***Swarm으로 배포하기***
  - todo-mysql.yml에 정의 된 서비스를 manager에서 ***todo_mysql 스택***으로 배포

```
$ docker  exec  -it  manager docker  stack  deploy  -c  /stack/todo-mysql.yml  todo_mysql
$ docker  exec  -it  manager docker  stack  ls
$ docker  exec  -it  manager docker  service  ls
$ docker  exec  -it  manager docker  service  ps  [Container ID]
```



Swarm을 이용한 실전 애플리케이션 개발 - MySQL 서비스 구축

- ***MySQL 컨테이너 확인 및 초기 데이터 저장***
  - 도커 컨테이너 안에 포함된 init-data.sh, tododb/sql 폴더의 SQL 파일 실행

```
$ docker exec -it manager \
        docker service ps todo_mysql_master --no-trunc --filter "desired-state=running"
$ docker exec -it [Node Container ID] \
        docker exec -it todo_mysql_master.1.[Service ID] bash
```

→ *접속 후 init-data.sh 실행*

```
$ docker exec -it manager docker service ps todo_mysql_master --no-trunc \
        --filter "desired-state=running" \
        -- format "docker exec -it {{.Node}} docker exec -it {{.Name}}.{{.ID}} bash"
```

→ *접속 후 init-data.sh 실행*

Swarm을 이용한 실전 애플리케이션 개발 - MySQL 서비스 구축

- *Master 컨테이너에서 init-data.sh 실행하여 테이블 및 초기 데이터 생성*

$ docker exec -it [Container ID] ***docker exec -it todo_mysql_master.1.[Service ID]*** \
init-data.sh

$ docker exec -it [Container ID] ***docker exec -it todo_mysql_master.1.[Service ID]*** \
mysql -ugihyo -pgihyo tododb

- *Slave 컨테이너에서 Master의 데이터가 반영 되었는지 확인*

$ docker exec -it manager ***docker service ps todo_mysql_slave*** --no-trunc \
--filter "desired-state=running" \
--format "docker exec -it {{.Node}} docker exec -it {{.Name}}.{{.ID}} bash"

$ docker exec -it [Container ID] ***docker exec -it todo_mysql_slave.1.***[Service ID] bash

mysql> select * from todo;

Swarm을 이용한 실전 애플리케이션 개발 – API 서비스 구축

- ***TODO 앱의 도메인 담당***
- ***Go 언어로 구현***
  - cmd/main.go 실행
    - MySQL 접속에 필요한 환경 변수 값 얻어오기
    - HTTP 요청 핸들러 생성 및 앤드포인트 등록, 서버 실행
  - env.go
    - 환경 변수 값을 받아 오는 코드
  - db.go
    - MySQL 접속 및 테이블 매핑
  - handler.go
    - 핸들러
    - TODO API의 요청 처리 → TodoHandler
      - ***serveGET*** → $ curl -s -XGET http://localhost:8080/todo?status=PROGRESS
      - ***servePOST*** → $ curl –XPOST -d '{"title": " 4장 집필하기", "content":"내용 " }' http://localhost:8080/todo
      - ***servePUT*** → $ curl –XPUT -d '{
        "id":1,"title": " 4장 집필하기",
        "content": "내용", "status":"PROGRESS"
        }' http://localhost:8080/todo

Swarm을 이용한 실전 애플리케이션 개발 - API 서비스 구축

- *API 서비스의 Dockerfile*

```
1  FROM golang:1.10
2
3  WORKDIR /
4  ENV GOPATH /go
5
6  COPY . /go/src/github.com/gihyodocker/todoapi
7  RUN go get github.com/go-sql-driver/mysql
8  RUN go get gopkg.in/gorp.v1
9  RUN cd /go/src/github.com/gihyodocker/todoapi && go build -o bin/todoapi cmd/main.go
10 RUN cd /go/src/github.com/gihyodocker/todoapi && cp bin/todoapi /usr/local/bin/
11
12 CMD ["todoapi"]
13
```

- ***API 서비스의 Dockerfile build***

$ docker build –t ***ch04/todoapi:latest*** .

$ docker tag ***ch04/todoapi:latest*** ***localhost:5000/ch04/todoapi:latest*** .

$ docker push ***localhost:5000/ch04/todoapi:latest***

Swarm을 이용한 실전 애플리케이션 개발 - API 서비스 구축

- *Swarm 에서 todoapi 서비스 실행하기*

```yaml
version: "3"
services:
  api:
    image: registry:5000/ch04/todoapi:latest
    deploy:
      replicas: 2
      placement:
        constraints: [node.role != manager]
    environment:
      TODO_BIND: ":8080"
      TODO_MASTER_URL: "gihyo:gihyo@tcp(todo_mysql_master:3306)/tododb?parseTime=true"
      TODO_SLAVE_URL: "gihyo:gihyo@tcp(todo_mysql_slave:3306)/tododb?parseTime=true"
    networks:
      - todoapp

networks:
  todoapp:
    external: true
```

Swarm을 이용한 실전 애플리케이션 개발 – API 서비스 구축

njone company

- ***Swarm 에서 todoapi 서비스 실행하기***

$ docker exec -it manager docker ***stack deploy -c*** /stack/todo-app.yml todo_app

$ docker exec -it manager docker ***service logs –f*** todo_app_api

- ***API 테스트***
  - todo_app_api 서비스가 어떤 Node에 설치 되었는지 정보 확인 → 해당 Node(Worker)로 접속
  - serveGET, servePOST, servePUT 테스트
  - ex) docker exec-it worker01 sh

    #/ docker ps

```
/ # docker ps
CONTAINER ID        IMAGE                                COMMAND                  CREATED             STATUS              PORTS                    NAMES
2ad3d860cd97        registry:5000/ch04/todoapi:latest    "todoapi"                29 minutes ago      Up 29 minutes                                todo_app_api.2.5cinawbrqe95rysi4j9yu3q94
dd6c1b419bc2        registry:5000/ch04/tododb:latest     "prehook add-server-…"   46 minutes ago      Up 46 minutes       3306/tcp, 33060/tcp      todo_mysql_master.1.2ll4gvfrallpa75gthow31ofc
/ # docker exec -it todo_app_api.2.5cinawbrqe95rysi4j9yu3q94 bash
root@2ad3d860cd97:/# curl -s -XGET http://localhost:8080/todo?status=DONE
[{"id":1,"title":"MySQL 도커 이미지 만들기","content":"MySQL 마스터와 슬레이브를 환경 변수로 설정할 수 있는 MySQL 이미지 생성","status":"DONE","created":"2020-01-04T01:38:31Z","updated":"2020-01-04T01:38:31Z"},{"id":2,"title":"MySQL 스택 만들기","content":"MySQL 마스터 및 슬레이브 서비스로 구성된 스택을 스웜 클러스터에 구축한다","status":"DONE","created":"2020-01-04T01:38:31Z","updated":"2020-01-04T01:38:31Z"}]root@2ad3d860cd97:/#
```

Swarm을 이용한 실전 애플리케이션 개발 – Nginx 서비스 구축

- *entrykit 템플릿을 이용하여 환경 변수 제어*

  {{ var "환경 변수명" }}

  {{ var "환경 변수명" | default "기본값" }}

  - *Nginx 설정 파일 → etc/nginx/nginx.conf.tmpl*
  - *백엔드 서버 지정 → etc/nginx/conf.d/upstream.conf.tml*
  - *라우팅 → etc/nginx/conf.d/public.conf.tml*

Swarm을 이용한 실전 애플리케이션 개발 – Nginx 서비스 구축

- ***API를 위한 Dockerfile***

```
1  FROM nginx:1.13
2
3  RUN apt-get update
4  RUN apt-get install -y wget
5  RUN wget https://github.com/progrium/entrykit/releases/download/v0.4.0/
   entrykit_0.4.0_linux_x86_64.tgz
6  RUN tar -xvzf entrykit_0.4.0_linux_x86_64.tgz
7  RUN rm entrykit_0.4.0_linux_x86_64.tgz
8  RUN mv entrykit /usr/local/bin/
9  RUN entrykit --symlink
10
11 RUN rm /etc/nginx/conf.d/*
12 COPY etc/nginx/nginx.conf.tmpl /etc/nginx/
13 COPY etc/nginx/conf.d/ /etc/nginx/conf.d/
14
15 ENTRYPOINT [ \
16   "render", \
17       "/etc/nginx/nginx.conf", \
18       "--", \
19   "render", \
20       "/etc/nginx/conf.d/upstream.conf", \
21       "--", \
22   "render", \
23       "/etc/nginx/conf.d/public.conf", \
24       "--" \
25 ]
26
27 CMD ["nginx", "-g", "daemon off;"]
```

$ docker build –t ***ch04/nginx:latest*** .

$ docker tag ***ch04/nginx:latest*** ***localhost:5000/ch04/nginx:latest*** .

$ docker push ***localhost:5000/ch04/nginx:latest***

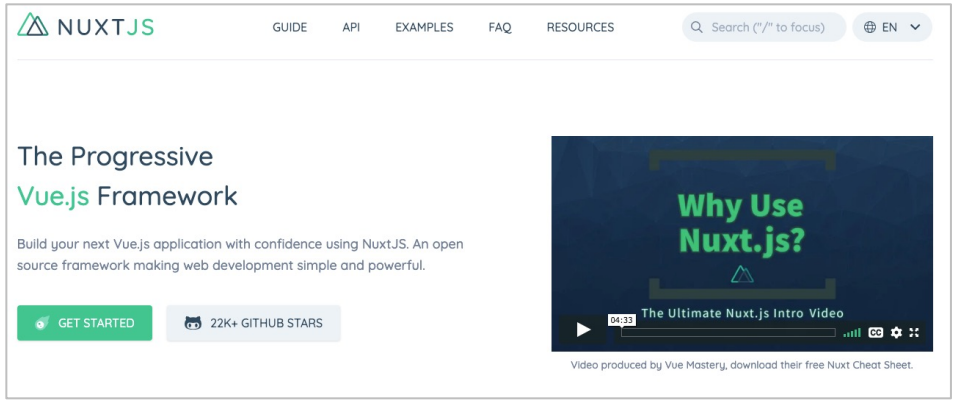
Swarm을 이용한 실전 애플리케이션 개발 – Nginx 서비스 구축

- *Nginx를 거쳐 API 서비스에 접근하기*
  - *조금 전에 작성한 todo_app_api 서비스 앞에 Nginx를 배치*

```yaml
version: "3"
services:
  nginx:
    image: registry:5000/ch04/nginx:latest
    deploy:
      replicas: 2
      placement:
        constraints: [node.role != manager]
    depends_on:
      - api
    environment:
      WORKER_PROCESSES: 2
      WORKER_CONNECTIONS: 1024
      KEEPALIVE_TIMEOUT: 65
      GZIP: "on"
      BACKEND_HOST: todo_app_api:8080
      BACKEND_MAX_FAILES: 3
      BACKEND_FAIL_TIMEOUT: 10s
      SERVER_PORT: 8000
      SERVER_NAME: todo_app_nginx
      LOG_STDOUT: "true"
    networks:
      - todoapp

  api:
    image: registry:5000/ch04/todoapi:latest
```

$ docker exec -it manager docker stack deploy -c ***/stack/todo-app.yml*** todo_app

- ***TODO의 진행 상황을 보여주는 웹 애플리케이션***
  - node.js
    - Vue.js 기반 프레임워크은 nuxt.js 사용



- ***TODO API 호출 및 페이지 HTML 렌더링 (todoweb/pages/index.vue)***

```
90  <script>
91  import axios from 'axios'
92  let todoApiUrl = process.env.TODO_API_URL || 'http://localhost:8000'
93  export default {
94    async asyncData (context) {
95      const { data: todoItems } = await axios.get(`${todoApiUrl}/todo?status=TODO`)
96      const { data: progressItems } = await axios.get(`${todoApiUrl}/todo?status=PROGRESS`)
97      const { data: doneItems } = await axios.get(`${todoApiUrl}/todo?status=DONE`)
```

Swarm을 이용한 실전 애플리케이션 개발 - Web 서비스 구축

- ***Web Service의 Dockerfile***

-	(todoweb) $ npm install	# 의존 모듈 다운로드
-	(todoweb) $ npm run build	# 릴리스용 빌드
-	(todoweb) $ npm run start	# 애플리케이션 서버 시작

→ Docker 이미지 생성 전 테스트

```
1  FROM node:9.2.0
2
3  WORKDIR /todoweb
4  COPY . /todoweb
5
6  RUN npm install -g vue-cli@2.9.3
7  RUN npm install
8  RUN npm run build
9
10 ENV HOST 0.0.0.0
11
12 CMD ["npm", "run", "start"]
13
14 EXPOSE 3000
```

$ docker build –t ***ch04/todoweb:latest*** .

$ docker tag ***ch04/todoweb:latest*** ***localhost:5000/ch04/todoweb:latest*** .

$ docker push ***localhost:5000/ch04/todoweb:latest***

- ***static file을 위한 Nginx proxy 설정***
  - 정적 파일은 웹 애플리케이션을 거치가 있고 Nginx에서 바로 응답
  - ex) /_nuxt/dist/vendor.XXXXX.js → http://HOST/_nuxt/vendor.XXXXX.js 로 접근

  - (todonginx) $ cp etc/nginx/conf.d/public.conf.tmpl ***etc/nginx/conf.d/nuxt.conf.tmpl***

```
 6      location /_nuxt/ {
 7          alias /var/www/_nuxt/$1;
 8          {{ if var "LOG_STDOUT" }}
 9          access_log  /dev/stdout json;
10          error_log   /dev/stderr;
11          {{ else }}
12          access_log  /var/log/nginx/assets_access.log json;
13          error_log   /var/log/nginx/assets_error.log;
14          {{ end }}
15      }
```

nuxt.conf.tmpl

Swarm을 이용한 실전 애플리케이션 개발 - Web 서비스 구축

- ***static file을 위한 Nginx proxy 설정***
  - todonginx/Dockerfile을 todonginx/Dockerfile-nuxt 파일로 복사
  - (todonginx) $ cp Dockerfile ***Dockerfile-nuxt***

```
12  COPY etc/nginx/nginx.conf.tmpl /etc/nginx/
13  COPY etc/nginx/conf.d/ /etc/nginx/conf.d/
14
15  ENTRYPOINT [ \
16    "render", \
17        "/etc/nginx/nginx.conf", \
18        "--", \
19    "render", \
20        "/etc/nginx/conf.d/upstream.conf", \
21        "--", \
22    "render", \
23        "/etc/nginx/conf.d/nuxt.conf", \
24        "--" \
25  ]
26
27  CMD ["nginx", "-g", "daemon off;"]
```

Dockerfile-nuxt

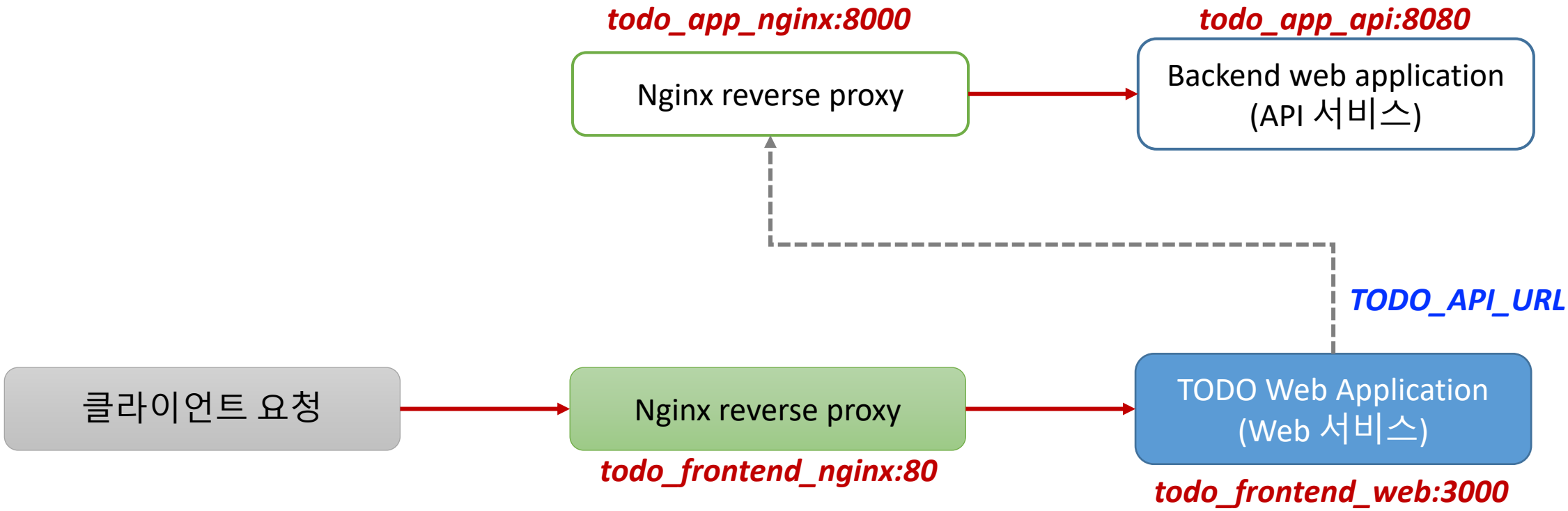
$ docker build -f ***Dockerfile-nuxt*** -t ***ch04/nginx-nuxt:latest*** .

$ docker tag ***ch04/nginx-nuxt:latest*** ***localhost:5000/ch04/nginx-nuxt:latest*** .

$ docker push ***localhost:5000/ch04/nginx-nuxt:latest***

public.conf → nuxt.conf

- ***Nginx를 통합 접근 허용***



- ***Nginx를 통합 접근 허용***

```yaml
version: "3"
services:
  nginx:
    image: registry:5000/ch04/nginx-nuxt:latest
    deploy:
      replicas: 2
      placement:
        constraints: [node.role != manager]
    depends_on:
      - web
    environment:
      SERVICE_PORTS: 80
      WORKER_PROCESSES: 2
      WORKER_CONNECTIONS: 1024
      KEEPALIVE_TIMEOUT: 65
      GZIP: "on"
      BACKEND_HOST: todo_frontend_web:3000
      BACKEND_MAX_FAILES: 3
      BACKEND_FAIL_TIMEOUT: 10s
      SERVER_PORT: 80
      SERVER_NAME: localhost
      LOG_STDOUT: "true"
    networks:
      - todoapp
    volumes:
      - assets:/var/www/_nuxt
```

```yaml
  web:
    image: registry:5000/ch04/todoweb:latest
    deploy:
      replicas: 2
      placement:
        constraints: [node.role != manager]
    environment:
      TODO_API_URL: http://todo_app_nginx:8000
    networks:
      - todoapp
    volumes:
      - assets:/todoweb/.nuxt/dist

networks:
  todoapp:
    external: true

volumes:
  assets:
    driver: local
```

$ docker exec -it manager docker stack deploy \
-c ***/stack/todo-frontend.yml*** todo_frontend

Swarm을 이용한 실전 애플리케이션 개발 - Ingress 서비스 구축

njone company

- *Ingress를 사용하여 TODO 웹 애플리케이션을 외부로 노출(공개)*

```yaml
version: "3"

services:
  haproxy:
    image: dockercloud/haproxy
    networks:
      - todoapp
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
    deploy:
      mode: global
      placement:
        constraints:
          - node.role == manager
    ports:
      - 80:80
      - 1936:1936 # for stats page (basic auth. stats:stats)

networks:
  todoapp:
    external: true
```

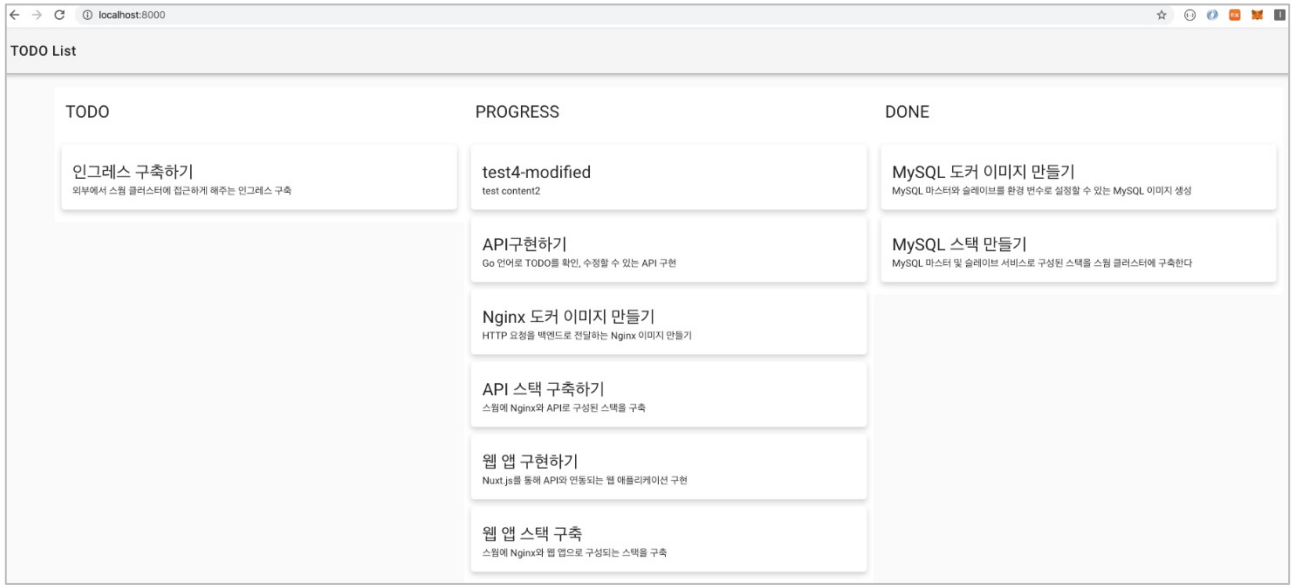
```
$ docker exec -it manager docker stack deploy \
        -c /stack/todo-ingress.yml todo_ingress
```

Swarm을 이용한 실전 애플리케이션 개발 - 실행

njone company

- *컨테이너 배치 구조 및 실행 화면*


```
curl -I http://localhost:8000
HTTP/1.1 200 OK
Server: nginx/1.13.12
Date: Sat, 04 Jan 2020 08:17:32 GMT
Content-Type: text/html; charset=utf-8
Content-Length: 19372
X-Powered-By: Express
ETag: "4bac-6EI9c5qpnD5C0jc2X5La7WL73sY"
Vary: Accept-Encoding
```



- *각 Node 별 Service(Container) 배치 구조 및 PORT 현황*

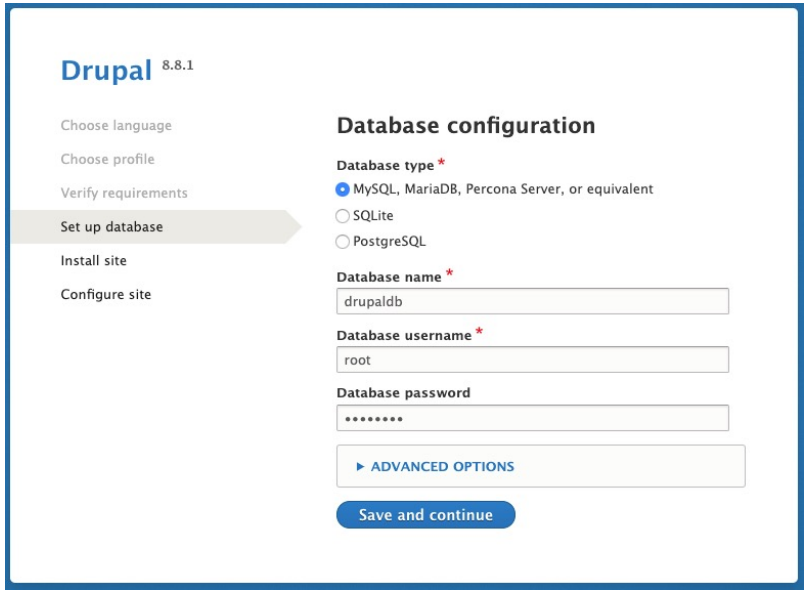
worker01			worker02			worker03		
todo_frontend_nginx	10.0.2.38	80				todo_frontend_nginx	10.0.2.37	80
todo_frontend_web	10.0.2.40	3000				todo_frontend_web	10.0.2.39	3000
			todo_app_nginx	10.0.2.15	8000	todo_app_nginx	10.0.2.14	8000
todo_app_api	10.0.2.12	8080	todo_app_api	10.0.2.11	8080			
todo_mysql_master	10.0.2.8	3306	todo_mysql_slave	10.0.2.4	3306	todo_mysql_slave	10.0.2.3	3306

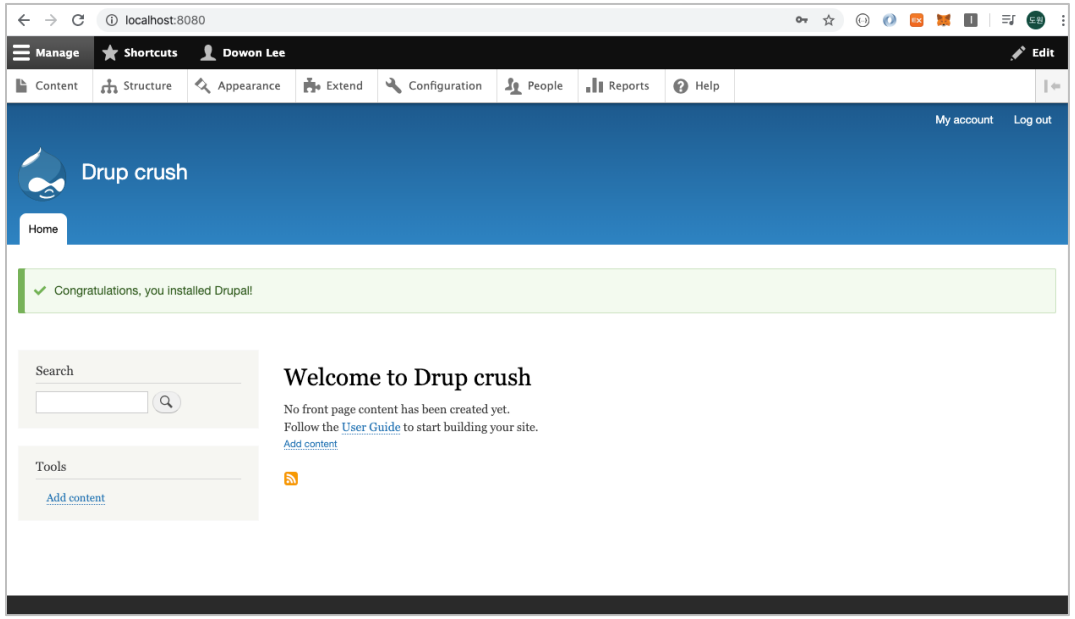
Docker Compose – LAB1

njone company

- Write Compose file for ***DRUPAL service (***https://www.drupal.com/)
  - Build a basic compose file for a ***Drupal content management system*** website. Docker Hub is your friend :)
  - Use the ***drupal image*** along with the ***mysql image***
  - Use ports to expose Drupal on ***8080*** so you can ***localhost:8080***
  - Be sure to set ***MYSQL_ROOT_PASSWORD*** for mysql
  - Walk though ***Drupal setup via browser***

  - Tip: Drupal assumes DB is localhost, but it's ***service name***
  - Extra Credit: Use ***volumes*** to store ***Drupal unique data***





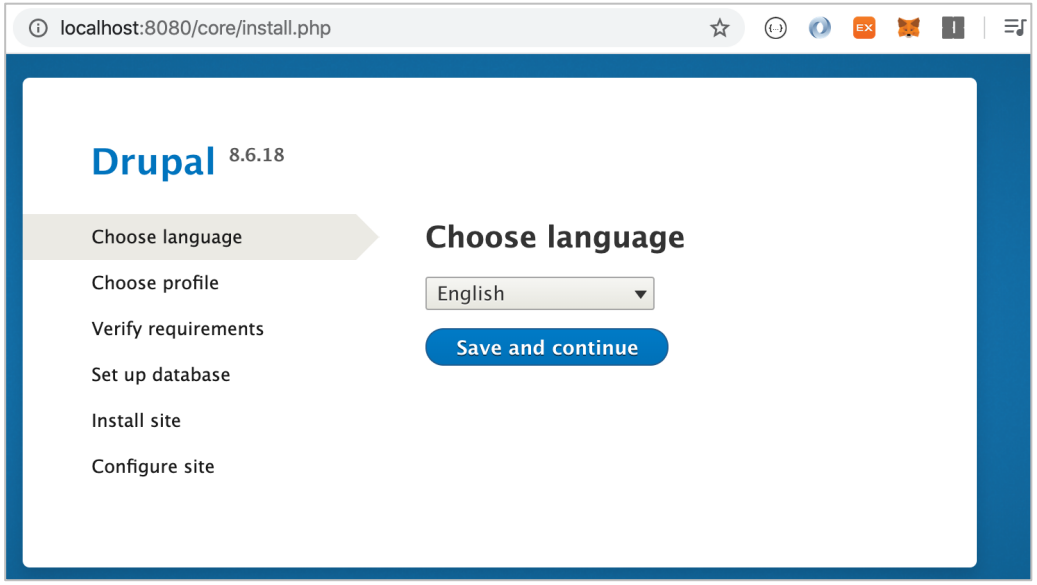
Docker Compose – LAB2

njone company

- Write Compose file with Dockerfile
  - Building ***custom drupa***l image for local testing
  - Start with Compose file from ***previous assignment*** (LAB1)
  - Make your ***Dockerfile*** and ***docker-compose.yml*** in dir compose-assignment-2
  - Use the ***drupal image*** along with the ***mysql image*** as before
  - Use README.md in that dir for details

```
▶ docker image ls
REPOSITORY                    TAG                 IMAGE ID            CREATED             SIZE
custom-drupal                 latest              84c2650021e3        6 hours ago         480MB
edowon0623/hello              latest              a9e48add24c9        2 days ago          163MB
```

```
 dowon@DOWON-MacBook  ~/Desktop/Work/docker/drupal/compose-assignment-2
▶ ls -al
total 24
drwxr-xr-x  5 dowon  staff   160  1 16 01:49 .
drwxr-xr-x  4 dowon  staff   128  1 16 01:34 ..
-rw-r--r--  1 dowon  staff   300  1 16 01:46 Dockerfile
-rw-r--r--  1 dowon  staff  3611  1 16 01:45 README.MD
-rw-r--r--  1 dowon  staff   602  1 16 01:55 docker-compose.yml
```



Docker Swarm – LAB3

njone company

- Create Voting App with multi services
  - Everything is using Docker Hub images, so no data needed on Swarm
  - Using Docker's Distributed ***Voting App***
  - use ***swarm-app-1*** directory in our course repo for
    - To set swarm, execute the following file in ***swarm-app-1*** folder.
    - ***docker-compose.yml***
      - We have already used this yaml file in Chapter 4 of our book.
  - requirements
    - We create a stack and services, you need to create a ***yaml file*** like ***todo_app.yml***
      - *volume x 1* (db-data,target=/var/lib/postgresql/data)
      - *networks x 2* (frontend, backend)
      - *services x 5* (voting-app, redis, db, worker, result-app)
      - *stack x 1* (my-vote-app)
  - Create the commands needed, spin up services, and test app
  - To vote, access the following URL
    - http://localhost:8000/
  - To see the results, access the following URL
    - http://localhost:5001/

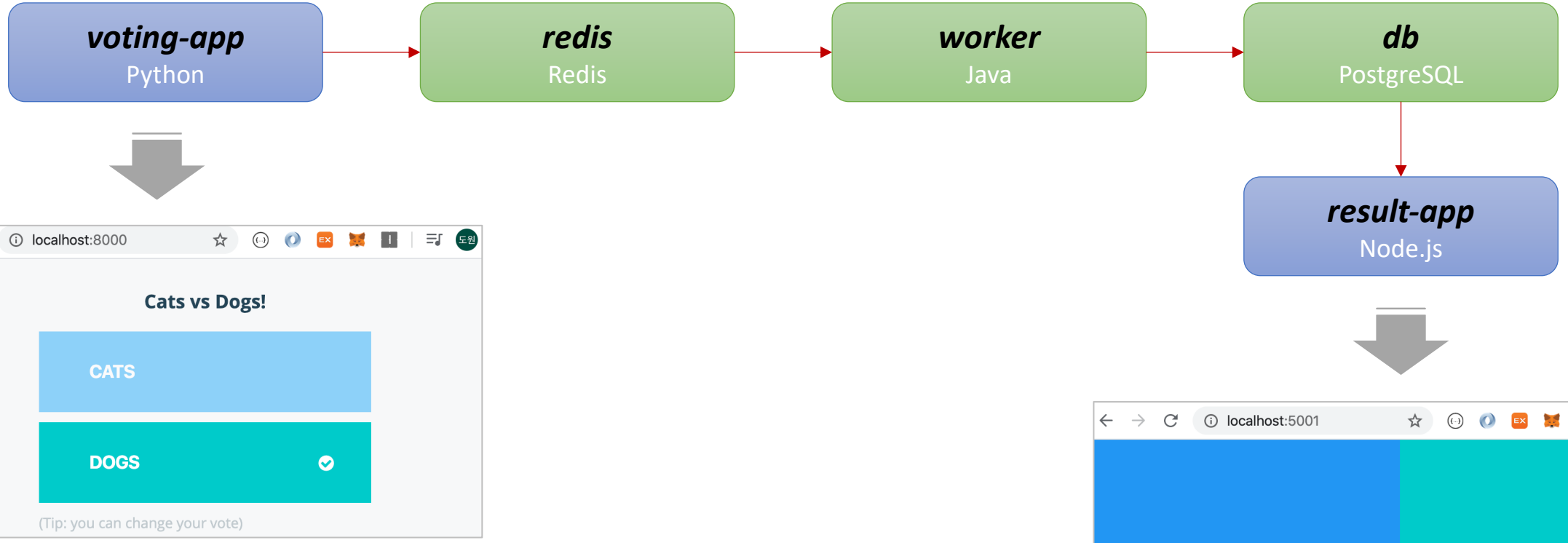
- bretfisher/examplevotingapp_vote
- bretfisher/examplevotingapp_worker:java
- bretfisher/examplevotingapp_result
- redis:3.2
- postgres:9.4

using images from https://hub.docker.com

Docker Swarm – LAB3

njone company

- Create Multi-Service App



Docker Swarm – LAB3

njone company

- Create Multi-Service App