

# Docker Hub와 Registry



## 도커 허브(Docker Hub)

- 도커 이미지 공개 저장소
- Root: 공식 저장소, official 마크
- User: repository:tag

## [실습2-1] Repo에 이미지 푸시



1.dockerhub 로그인/ repository를 만든다.

- repo name: docker-whale

Repository Name: k16wire

Visibility: private

Create

[https://docs.docker.com/engine/getstarted/step\\_four/](https://docs.docker.com/engine/getstarted/step_four/)

\$ docker tag 85e9c577cf0c k16wire/docker-whale:latest

이미지 ID      DockerHub 계정 이름      이미지 이름      태그



2.도커허브 로그인

```
$ docker login
```

3.이미지에 repo 정보를 태그한다.

```
$ docker tag 85e9c577cf0c k16wire/docker-whale:latest
```

4.이미지를 repo에 푸시한다.

```
$ docker push k16wire/docker-whale:latest
```

[https://docs.docker.com/engine/getstarted/step\\_four/](https://docs.docker.com/engine/getstarted/step_four/)

## 레지스트리(Registry)

- 도커 이미지를 저장하고 공유할수 있는 서버

- 오픈소스, Apache 라이선스

- v1과 v2가 호환되지 않는다.

- 클라우드: DockerHub

- 인트라넷: DTR

파키  
Singp  
Layer

← GoLang  
←

## [실습2-2] Private Registry 실행



registry 컨테이너 실행하기

```
$ docker run -d -p 5000:5000 --name myregistry  
registry:2
```

## [실습2-3] Private Registry 푸시



이미지에 Private Repo 태그하기

```
$ docker tag ae7d6ffe6742 localhost:5000/docker-  
whale:latest
```

이미지 푸시

```
$ docker push localhost:5000/docker-whale:latest
```



### 1. 기존 이미지 삭제

```
$ docker rmi -f 7d9495d03763
```

### 2. 이미지 풀링

```
$ docker pull localhost:5000/docker-whale:latest
```

### 3. 이미지 실행

```
$ docker run localhost:5000/docker-whale:latest
```

→ "저장소를 따로 써라" ⇒ Host Volume  
NAS

## 레지스트리 - 저장소

- 이미지 데이터를 어디에 저장할것인가?

- Local

```
$ docker run -d -p 5000:5000 -v $(pwd)/registry-data:/var/lib/registry --name myregistry registry:2
```

- Storage Drivers:

- <https://docs.docker.com/registry/storage-drivers/>



# 레지스트리 - 저장소

- Storage Drivers(<https://docs.docker.com/registry/storage-drivers>)
  - s3: AWS S3 bucket
  - azure: Microsoft Blob Storage
  - swift: Openstack Swift
  - oss: Aliyn OSS
  - gcs: Google Cloud Storage



계정자원에 연결

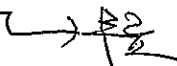


## 레지스트리 - 보안

- TLS, 도메인 지원

\* Image 보안 공략 '19년 초

- Let's Encrypt 추천



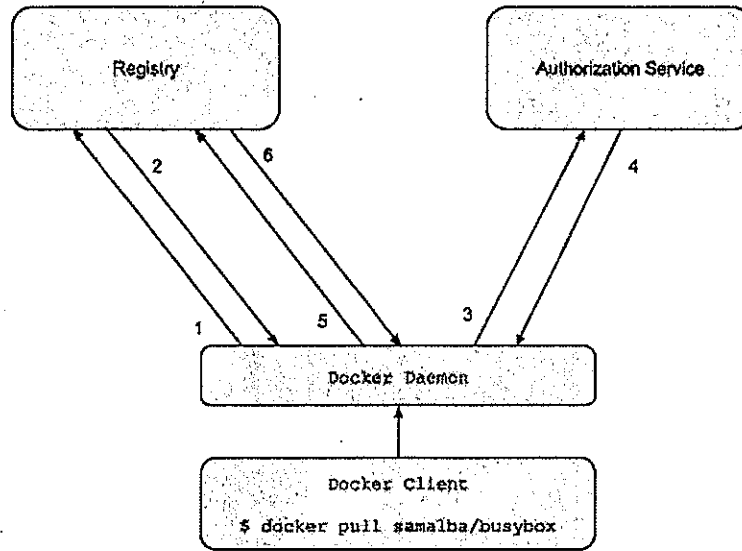
- 접근 권한 관리

- 인증: nginx를 활용한 basic auth

- 이미지 무결성: ssh 사이닝



# 레지스트리 보안 모델 참고



<https://docs.docker.com/registry/spec/auth/token/>

17

## 레지스트리 - 팁

- 네이밍 기준: 루트, Repo 명 ⇒ 이미지 배포 → 확실히 관리 이슈
- 이미지 데이터 관리: gc 필요 ⇒ 기존 이미지 관리
- 성능 이슈 ⇒ 일반적 수백 개  
다들 모르는 것도 있음

18



1.registry.yml 작성

2.docker-compose로 registry 실행

```
$ docker-compose -f registry.yml up -d
```

## registry.yml

```
registry:
  restart: always
  image: registry:2
  ports:
    - 5000:5000
  environment:
    REGISTRY_HTTP_TLS_CERTIFICATE: /certs/domain.crt
    REGISTRY_HTTP_TLS_KEY: /certs/domain.key
    REGISTRY_AUTH: htpasswd
    REGISTRY_AUTH_HTPASSWD_PATH: /auth/htpasswd
    REGISTRY_AUTH_HTPASSWD_REALM: Registry Realm
  volumes:
    - /path/data:/var/lib/registry
    - /path/certs:/certs
    - /path/auth:/auth
```



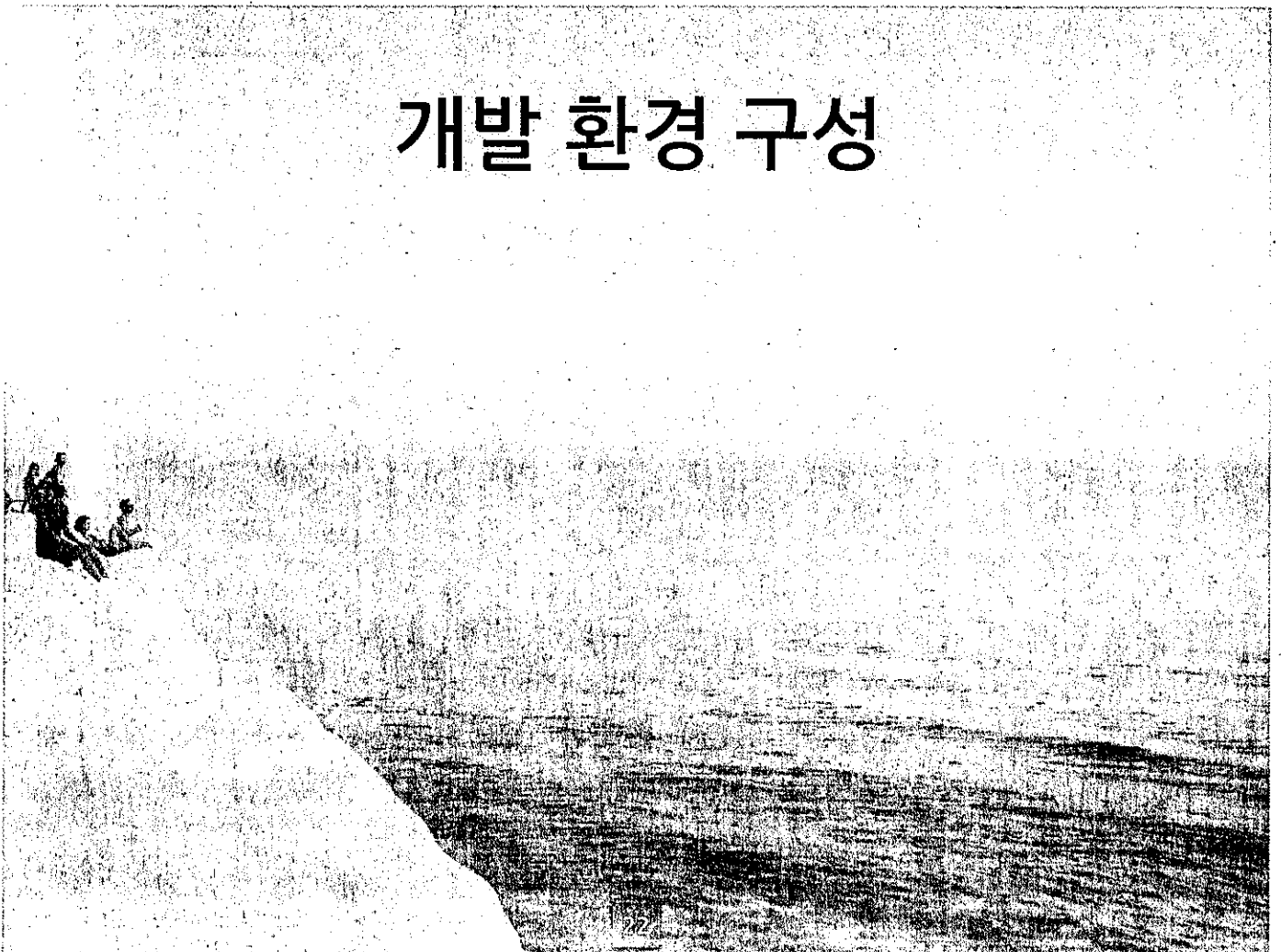
# registry.yml

```
storage: s3
s3_access_key: <S3_ACCESS_KEY>
s3_secret_key: <S3_SECRET_KEY>
s3_bucket: docker-registry
s3_encrypt: true
s3_secure: true
s3_region: ap-northeast-1
secret_key: <SECRET_KEY>
storage_path: /images
```

<https://dobest.io/docker-registry-02-install-on-ec2-and-s3/>

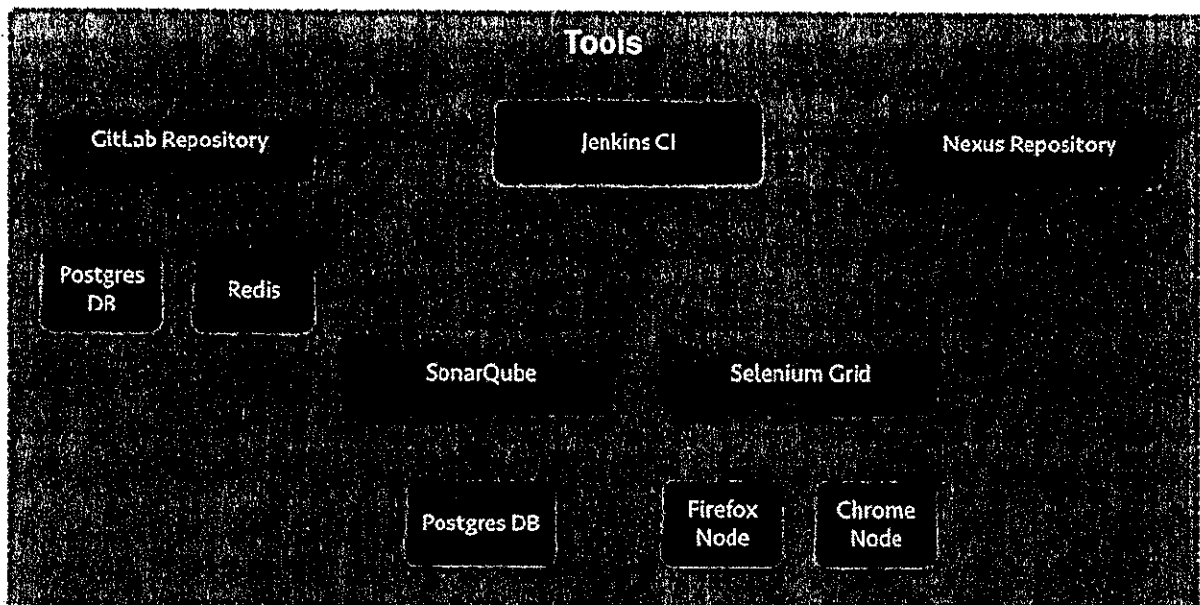


## 개발 환경 구성



# 개발에 필요한 도구를 도커로 구성하자

## Docker CI Tool Stack



<https://github.com/marcelbirkner/docker-ci-tool-stack>



## [실습2-6] docker-ci-tool-stack



1. docker-ci 머신을 생성한다.

```
$ docker-machine create -d virtualbox docker-ci
```

2. docker-ci 머신을 연결한다.

```
$ eval $(docker-machine env docker-ci)
```

3. github 레파지터리 연결

```
$ git clone https://github.com/marcelbirkner/  
docker-ci-tool-stack.git
```

4. tool-stack 실행

```
$ docker-compose up -d
```

## [실습2-6] docker-ci-tool-stack



5. 각 서버에 접속한다.

| Tool          |   |
|---------------|---|
| Jenkins       | <a href="http://192.168.99.106:18080/">http://192.168.99.106:18080/</a>                       |
| SonarQube     | <a href="http://192.168.99.106:19000/">http://192.168.99.106:19000/</a>                       |
| Nexus         | <a href="http://192.168.99.106:18081/nexus">http://192.168.99.106:18081/nexus</a>             |
| GitLab        | <a href="http://192.168.99.106/">http://192.168.99.106/</a>                                   |
| Selenium Grid | <a href="http://192.168.99.106:4444/grid/console">http://192.168.99.106:4444/grid/console</a> |

*docker-compose*

# 개발을 위한 도구들

| 구분    | 제품       | 비고  |
|-------|----------|---|
| 형상관리  | git-scm  | <a href="https://git-scm.com/">https://git-scm.com/</a>                           |
|       | gitlab   | <a href="https://about.gitlab.com/">https://about.gitlab.com/</a>                 |
|       | gogs     | <a href="https://gogs.io/">https://gogs.io/</a>                                   |
|       | Yona     | <a href="http://yona.io/">http://yona.io/</a>                                     |
| 빌드    | Jenkins  | <a href="https://jenkins.io/">https://jenkins.io/</a>                             |
|       | Drone    | <a href="https://github.com/drone/drone">https://github.com/drone/drone</a>       |
|       | go       | <a href="https://www.gocd.io/">https://www.gocd.io/</a>                           |
| 패키지관리 | Nexus    | <a href="http://www.sonatype.org/nexus/">http://www.sonatype.org/nexus/</a>       |
| 정적분석  | SonaQube | <a href="https://www.sonarqube.org/">https://www.sonarqube.org/</a>               |
| 성능테스트 | nGrinder | <a href="https://naver.github.io/ngrinder/">https://naver.github.io/ngrinder/</a> |



# 개발을 위한 도구들

| 구분   | 제품        | 비고  |
|------|-----------|---|
| 이슈관리 | Redmine   | <a href="http://www.redmine.org/">http://www.redmine.org/</a>                                   |
|      | trac      | <a href="https://trac.edgewall.org/">https://trac.edgewall.org/</a>                             |
|      | mantis    | <a href="http://www.mantisbt.org/">http://www.mantisbt.org/</a>                                 |
| 위키   | MediaWiki | <a href="https://www.mediawiki.org/wiki/MediaWiki">https://www.mediawiki.org/wiki/MediaWiki</a> |
|      | DokuWiki  | <a href="https://www.dokuwiki.org/dokuwiki#">https://www.dokuwiki.org/dokuwiki#</a>             |

<https://opensource.com/business/16/2/top-issue-support-and-bug-tracking-tools>



# GitLab

- GitLab은 이슈관리, 코드리뷰, CI/CD를 지원하는 통합 개발환경 서버
- 라이선스: CM 버전은 무료, 엔터프라이즈 유료, 클라우드(무료, 유료)



29

# GitLab

- 데이터 저장위치

| 컨테이너            | 설명            |
|-----------------|---------------|
| /var/opt/gitlab | 어플리케이션 데이터 저장 |
| /var/log/gitlab | 로그 저장         |
| /etc/gitlab     | 설정파일 저장       |

- 포트: 80,443,22

<https://docs.gitlab.com/omnibus/docker/#prerequisites>

30

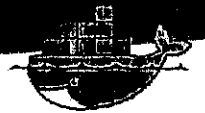
## [실습2-7] git-lab 설치 및 실행



1. gitlab-ce를 실행한다.

```
$ docker run --detach \  
  --hostname gitlab.example.com \  
  --publish 443:443 --publish 80:80 --publish  
22:22 \  
  --name gitlab \  
  --restart always \  
  --volume /srv/gitlab/config:/etc/gitlab \  
  --volume /srv/gitlab/logs:/var/log/gitlab \  
  --volume /srv/gitlab/data:/var/opt/gitlab \  
  gitlab/gitlab-ce:latest
```

## [실습2-8] docker-compose로 gitlab 실행



1. gitlab.yml을 작성한다.
2. gitlab-ce를 docker-compose로 실행한다.

```
$ docker-compose -f gitlab.yml up -d
```

# gitlab.yml

```
web:
  image: 'gitlab/gitlab-ce:latest'
  restart: always
  hostname: 'gitlab.example.com'
  environment:
    GITLAB_OMNIBUS_CONFIG: |
      external_url 'https://gitlab.example.com'
  ports:
    - '80:80'
    - '443:443'
    - '22:22'
  volumes:
    - '/srv/gitlab/config:/etc/gitlab'
    - '/srv/gitlab/logs:/var/log/gitlab'
    - '/srv/gitlab/data:/var/opt/gitlab'
```

<https://gitlab.com/gitlab-org/omnibus-gitlab/blob/master/docker/docker-compose.yml>

GitLab이 업그레이드 되면  
어떻게 하나요?

\* 자습지 ⇒ Data는 외부

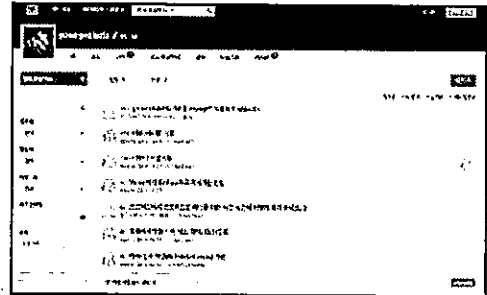
# Yona

- 프로젝트 단위로 형상관리, 이슈관리, 코드리뷰, 게시판을 지원하는 협업개발 플랫폼

- 네이버 Yobi의 후속 프로젝트

- 참고자료

↳ Naver 개발환경 표준



- <https://github.com/yona-projects/yona>
- <https://hub.docker.com/r/yongseoklee/docker-yona/>



## [실습2-9] yona 설치 및 실행



1. yona를 설치한다.

```
$ docker run --d \  
  --publish 9000:9000 \  
  --name yona \  
  --restart always \  
  yongseoklee/docker-yona:latest
```



# gogs

- go로 개발된 git 서비스, 가볍고 빠르다.
- 웹콘솔 지원



37

## [실습2-10] gogs 설치 및 실행



### 1. data 컨테이너 생성

```
$ docker run --name=gogs-data --entrypoint /bin/true gogs/gogs
```








### 2. gogs 컨테이너 생성

```
$ docker run -d --name=gogs --volumes-from gogs-data -p 1022:22 -p 3000:3000 gogs/gogs
```

<https://blog.asamaru.net/2015/09/21/how-to-install-gogs-on-centos/>

# Nexus Repo. OSS

- 다양한 형식의 컴포넌트 레파지터리 매니저
- <https://www.sonatype.com/nexus-repository-oss>

| Component Format   | Nexus Repository OSS | Artifactory OSS |
|--|----------------------|-----------------|
|  Java   | FREE                 | FREE            |
|  Maven  | FREE                 | ---             |
|  Gradle | FREE                 | ---             |
|  SBT    | FREE                 | ---             |
|  GPG    | FREE                 | ---             |
|  P2     | FREE                 | ---             |
|  PyPI | FREE                 | ---             |



## [실습2-11] nexus 설치 및 실행



### 1. data 컨테이너 생성

```
$ docker run -d --name nexus-data sonatype/nexus  
echo "data-only container for Nexus"
```

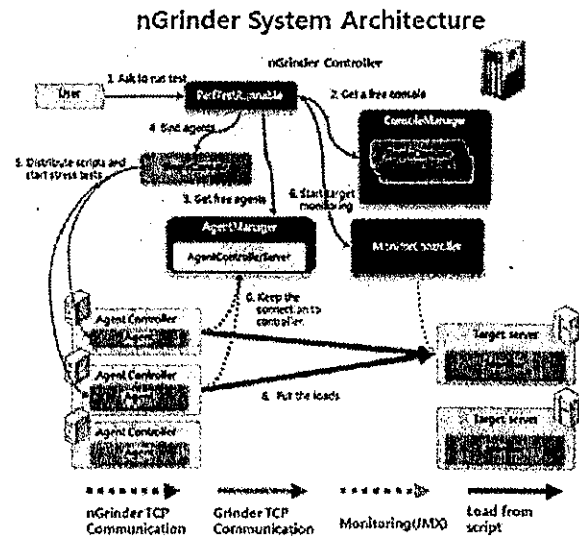
### 2. nexus 컨테이너 생성

```
$ docker run -d -p 8081:8081 --name nexus --  
volumes-from nexus-data sonatype/nexus
```

### 3. 웹콘솔 (기본계정 admin/admin123)

# nGrinder

- 서버에 대한 성능 테스트를 위한 오픈소스
- 성능테스트를 위한 웹UI
- Agent와 Controller



41

## [실습2-12] ngrinder 컨트롤러 설치

### 1. ngrinder 컨트롤러 생성

```
$ docker run -d -v ~/.ngrinder:/root/.ngrinder -p 80:80 -p 16001:16001 -p 12000-12009:12000-12009 ngrinder/controller:3.3
```

### 2. ngrinder UI 접속

<http://192.168.99.100> (admin/admin)

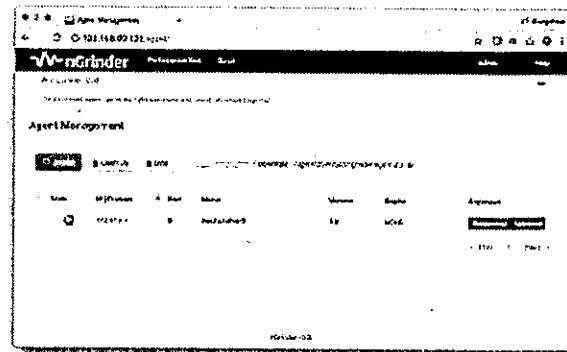


### 1. ngrinder 에이전트 생성

```
$ docker run -d -e  
'CONTROLLER_ADDR=controller_ip:80' ngrinder/  
agent:3.3
```

### 2. 컨트롤러 에이전트 확인

<http://192.168.99.102/agent/>



## Redmine

- 프로젝트 관리, 이슈관리 도구, 위키
- RoR 로 개발
- 형상서버 연계, 현황 차트 제공



## [실습2-14] Redmine 설치 및 실행



### 1. SQLite3기반 설치

```
$ docker run -d -P --name myredmine redmine
```

### 2. PostgreSQL 기반 설치

```
$ docker run -d --name redmine-postgres -e  
POSTGRES_PASSWORD='password1!' -e  
POSTGRES_USER=redmine postgres
```

```
$ docker run -d -p 3000:3000 --name myredmine --  
link redmine-postgres:postgres redmine
```

## [실습2-15] Compose로 Redmine 설치



```
version: '2'  
services:  
  redmine:  
    image: redmine  
    ports:  
      - 8080:3000  
    environment:  
      REDMINE_DB_MYSQL: db  
      REDMINE_DB_PASSWORD: example  
    depends_on:  
      - db  
    restart: always  
  db:  
    image: mariadb  
    environment:  
      MYSQL_ROOT_PASSWORD: example  
      MYSQL_DATABASE: redmine  
    restart: always
```

# MediaWiki

- 위키피디아 같은 오픈소스 위키
- PHP로 개발



## [실습2-16] MediaWiki 설치 및 실행



### 1. MySQL 설치

```
$ docker run -d --name mediawiki-mysql \  
-e MYSQL_ROOT_PASSWORD='password1' mysql
```

```
$ docker run --name my-mediawiki \  
--link mediawiki-mysql:mysql \  
-p 8080:80 -d synctree/mediawiki
```

### 2. 접속

<http://localhost:8080>

# LocalSettings.php 파일을 어떻게 위키에 적용하나요?

<https://github.com/besnik/tutorials/tree/master/docker-mediawiki>

## [실습2-17] MediaWiki 로컬 설정 반영



### 1. LocalSetting.php 로 설정추가

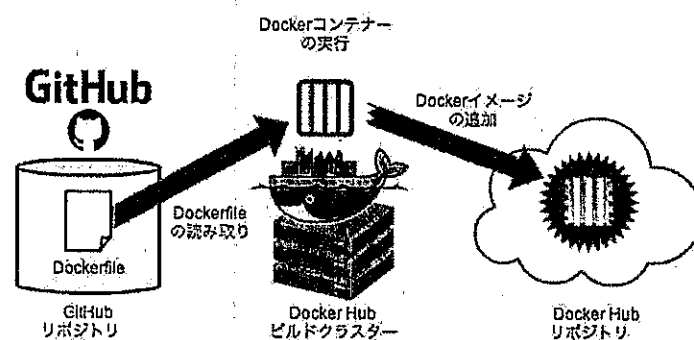
```
$ docker run -d --name my-mediawiki \  
--link mediawiki-mysql:mysql \  
-p 8080:80 -v /home/docker/mediawiki/  
LocalSettings.php:/var/www/html/LocalSettings.php  
synctree/mediawiki
```

# Docker CI 환경 구성



## github & DockerHub

github 과 DockerHub를 이용한 자동빌드 환경 구성

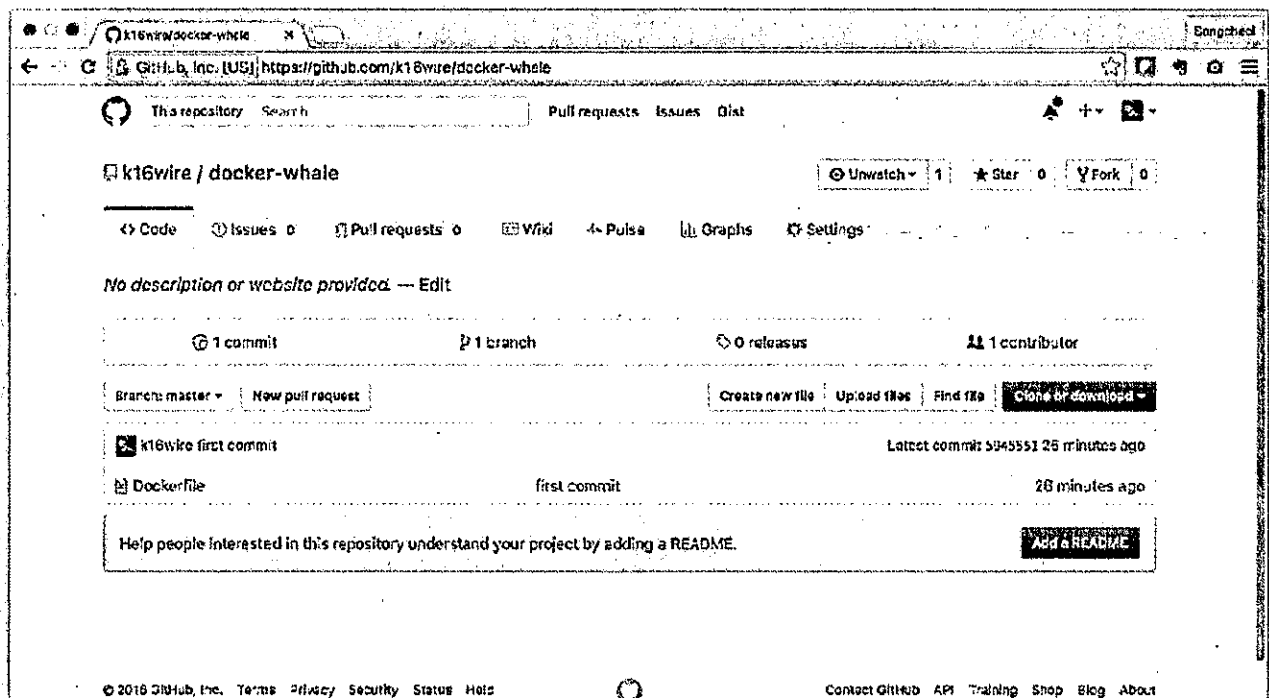


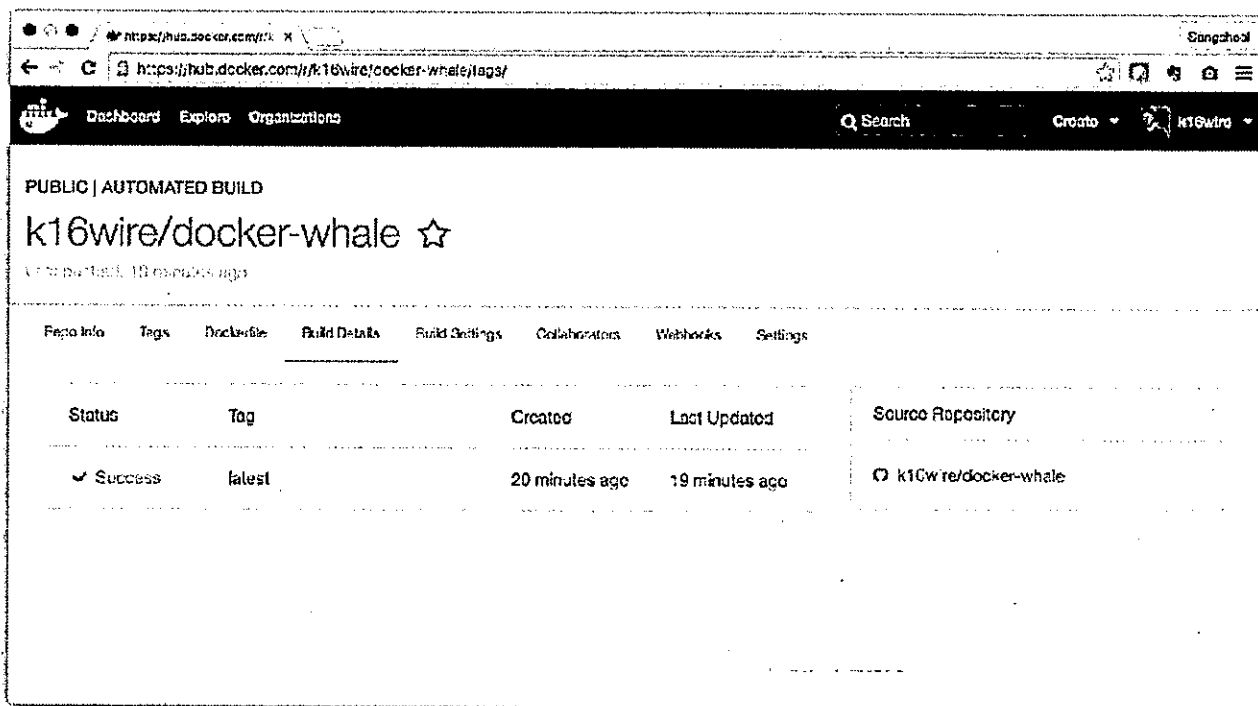


## [실습2-18] DockerHub Automated Build



1. github 에 도커 이미지 빌드를 위한 소스파일 추가
2. DockerHub에서 'Create Automated Build'
3. github 레파지터리 연결
4. 빌드 트리거 실행



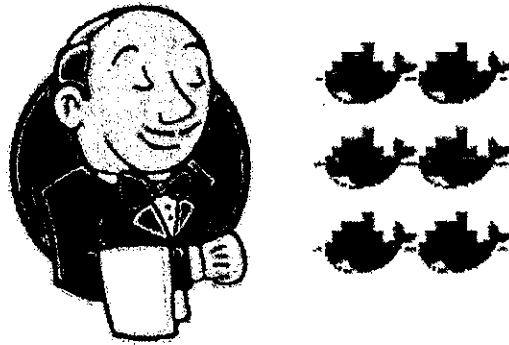


## Jenkins & Docker

- US1: 도커 이미지를 빌드/테스트/배포하는데 Jenkins를 활용한다.
- US2: Jenkins Master & Slave를 도커로 실행한다.

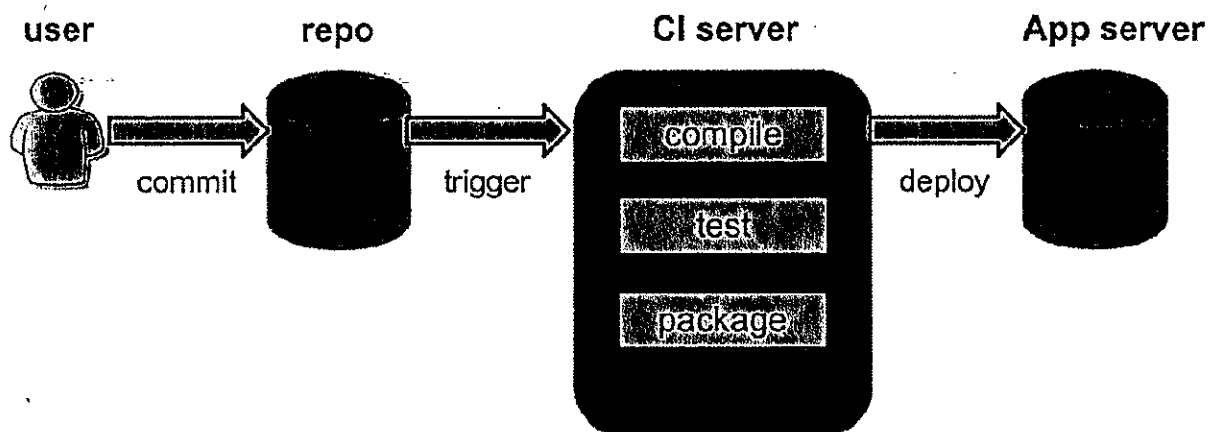


# US1:도커 이미지 빌드



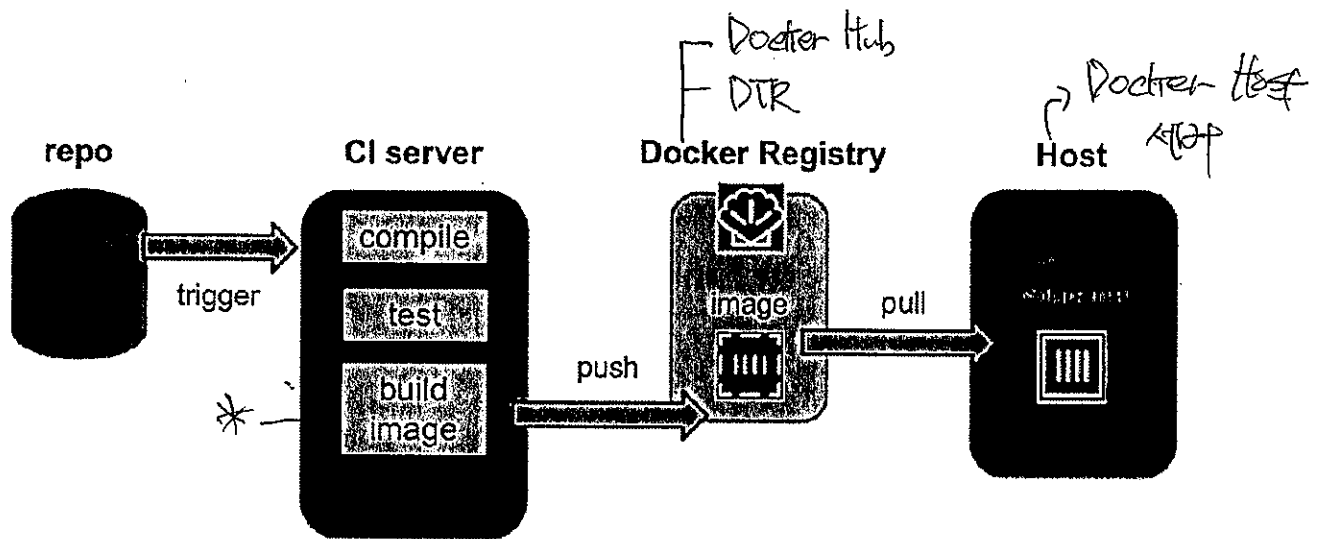
57

## 일반 CI 환경



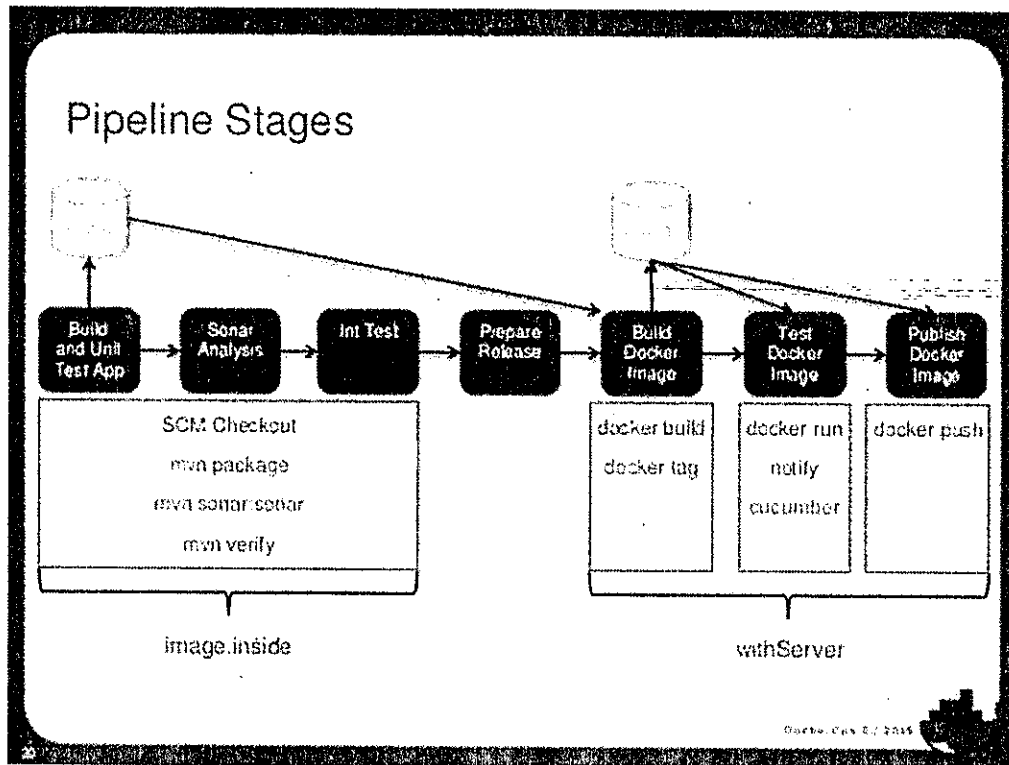
58

# Docker CI 환경



\* 이미지 빌드 Host에서

59



# Docker in Docker

- Docker 컨테이너가 Docker 명령어를 실행할 수 있게 하려면

- Docker 소켓 파일에 대한 접근권한 *→ docker.sock*

- Docker 클라이언트, 실행권한

*↳ 클라이언트/서버간의 소켓파일 공유*

<http://pragmaticstory.com/2015/07/02/jenkins%EB%A1%9C-docker-%EC%9D%B4%EB%AF%B8%EC%A7%80-%EB%A7%8C%EB%93%A4%EA%B8%B0/>



## [실습2-19] Image 빌드용 Jenkins 실행



1. docker 이미지 빌드용 jenkins 컨테이너 실행

```
$ docker run -d --name jenkins -p 8080:8080 -v /var/run/docker.sock:/var/run/docker.sock k16wire/docker-jenkins
```

2. jenkins 암호 입력

```
$ docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

3. jenkins 콘솔 접속

*↳ Jenkins Plugin*  
*↳ 컨테이너 내부 명령어 실행*

## [실습2-20] Image 빌드용 Jenkins 실행



docker-compose로 이미지 빌드용 jenkins 컨테이너 실행

```
jenkins:
  image: k16wire/docker-jenkins:latest
  ports:
    - 8080:8080
  volumes:
    - /data/jenkins:/var/jenkins_home
    - /var/run/docker.sock:/var/run/docker.sock
```

## [실습2-21] Image 빌드 Job

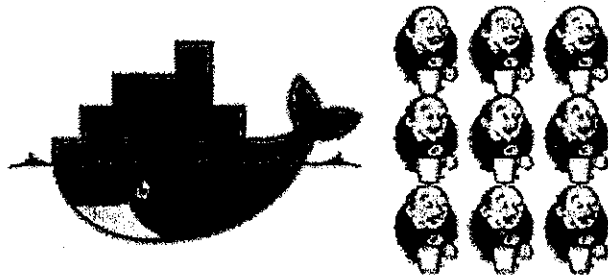


1. jenkins에 이미지를 빌드하는 Job을 추가한다.

- parameter: String: TAG
- git: <https://github.com/k16wire/docker-whale.git>
- shell script

```
docker build -t docker-whale .
docker tag docker-whale k16wire/docker-whale:${TAG}
docker login -e 이메일 -u 아이디 -p 암호
docker push k16wire/docker-whale:${TAG}
```

# US2:Jenkins Master & Slave



65

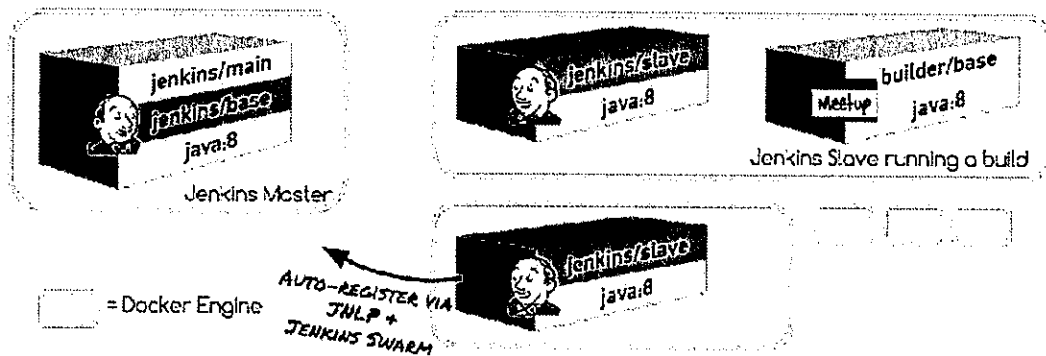
## Jenkins Master/Slave

Master/Slave 구성의 장점

- 다양한 환경의 빌드 진행이 가능
- 오래걸리는 빌드를 슬레이브에 위임 가능
- Jenkins를 여러개 사용하는것에 비해 관리가 용이

66

# Jenkins Master/Slave



<https://www.youtube.com/watch?v=PFCSSiT-UUQ>

<http://pragmaticstory.com/?p=219>

<https://www.youtube.com/watch?v=YYG8DOE8Pco>



## [실습2-22] slave 빌드



1. Jenkins에 Docker plugin 설치
2. Jenkins에 cloud 영역을 설정한다.
3. Docker Template을 추가한다.
4. Slave가 빌드하는 Job을 정의한다.

플러그인 버그 해결전까지 실습을 미룹니다.

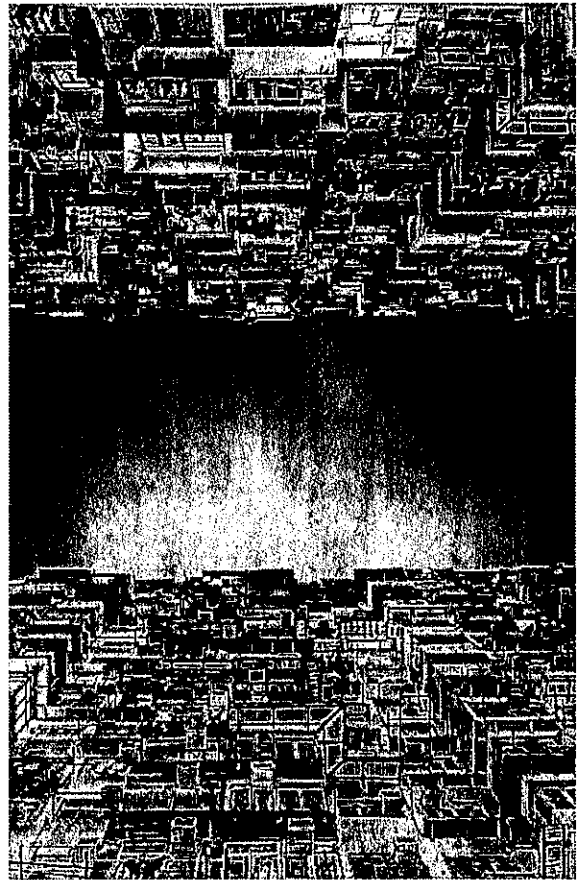


# Docker 멀티 호스트 구성

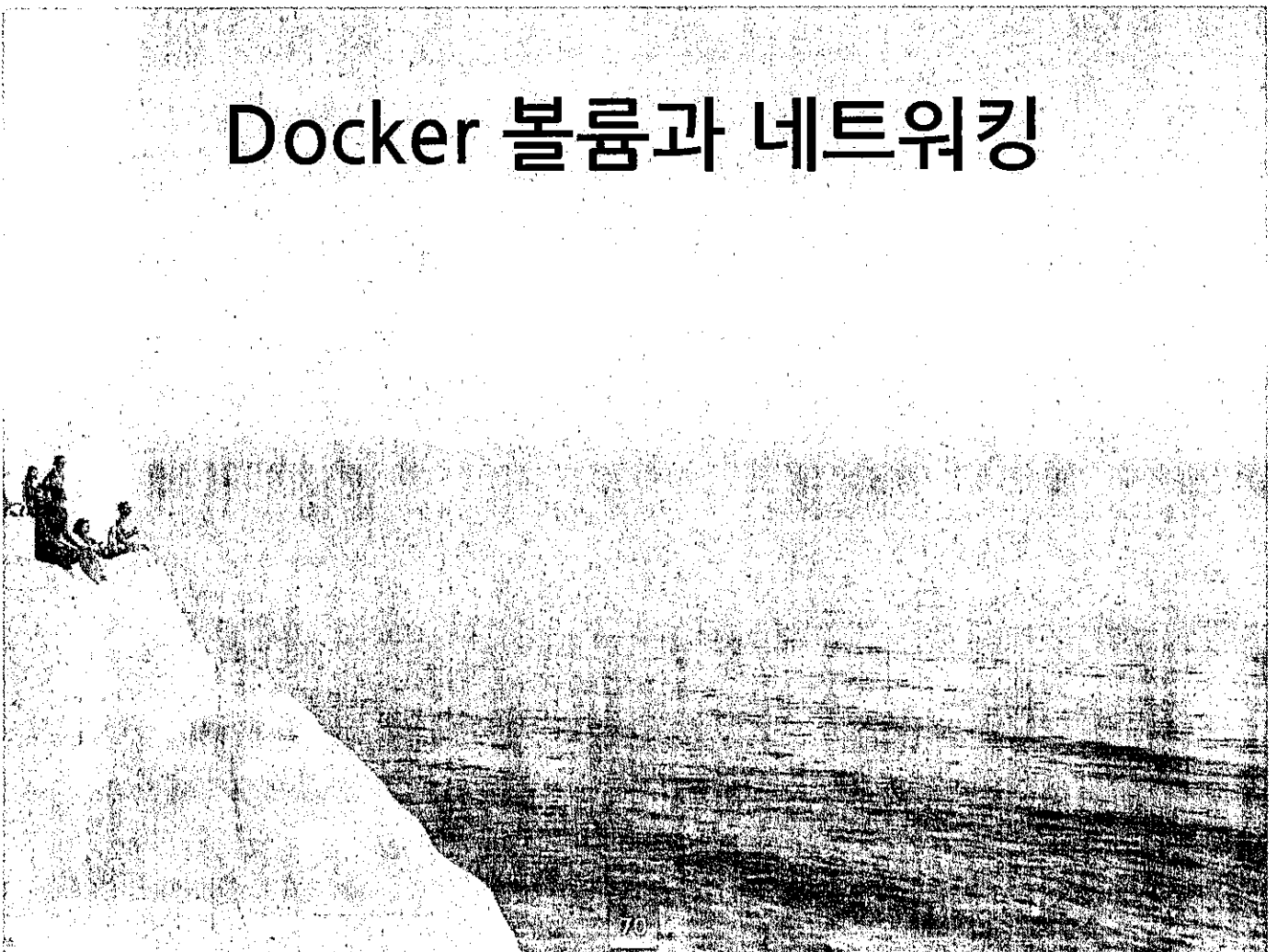
Docker 볼륨과 네트워킹

Docker Swarm

Docker Security



## Docker 볼륨과 네트워킹



# 컨테이너 볼륨 Container Volume

## Volume 장점

- 컨테이너와 데이터 분리
- 컨테이너간 데이터 공유
- I/O 성능 향상  $\Rightarrow$  Docker Engine 은 거쳐 않고 직접 Host 용
- 호스트와 컨테이너간 파일 공유

# Volume 관리 유형

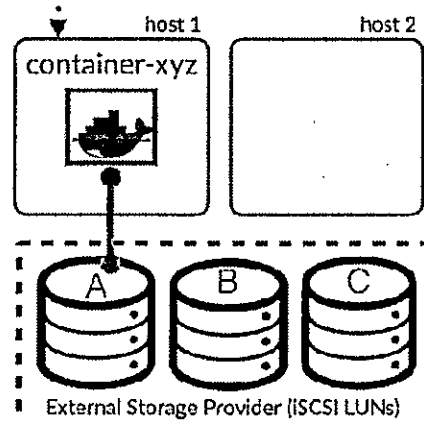
- 케이스1: 컨테이너 내부에 저장한다. — *일시적인*
- 케이스2: 도커 UFS에 저장한다. — *Docker Daemon 관리영역*  
— *Docker의 삭제작업 실행*  
— *작은사이즈*
- 케이스3: 도커 호스트 파일 시스템 볼륨 마운트
- 케이스4: volume-driver를 이용해 네트워크로 연결된 장치에 저장한다.

73

## 케이스4: 외부 Volume이 필요할까

Create and Run Container on Host 1

```
$ docker run -d -v /host/data:/data --volume-driver=flocker  
--name=container-xyz app
```

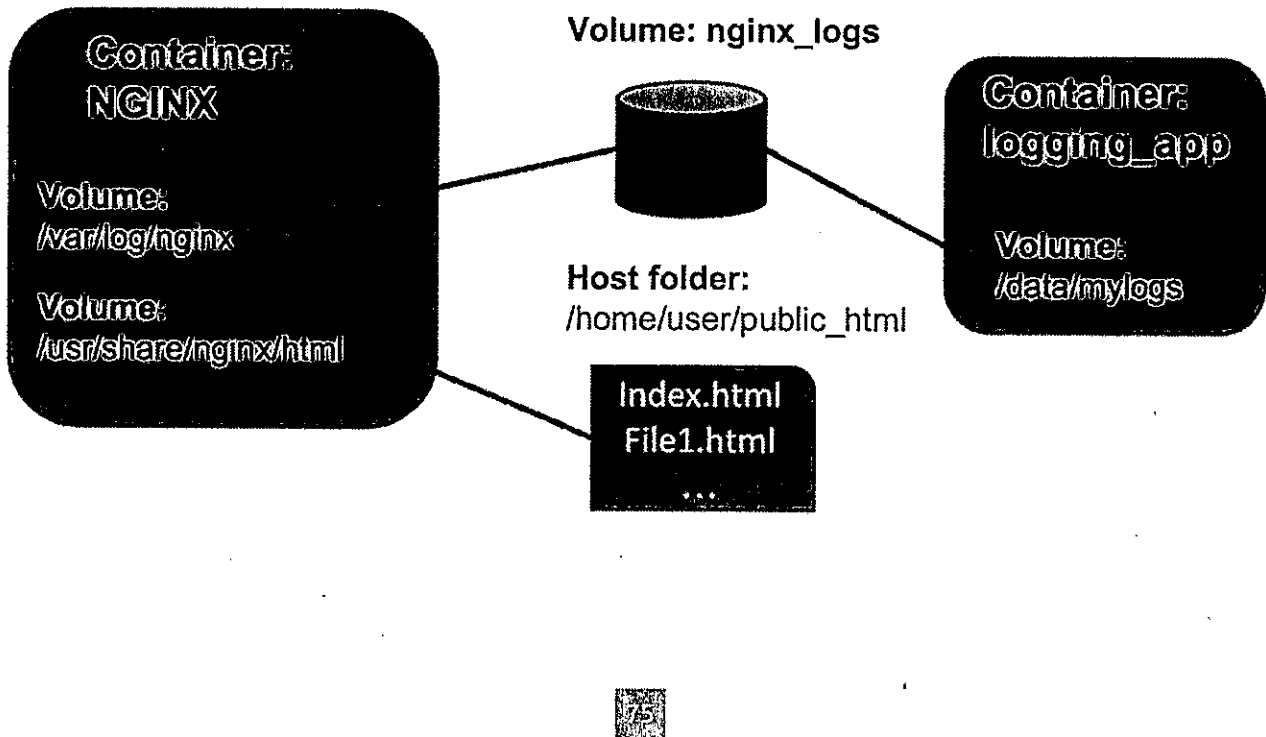


1/4

<https://clusterhq.com/2015/12/09/difference-docker-volumes-flocker-volumes/>

74

# 컨테이너 Volume 공유 사례



## Volume 주요 명령어

- docker volume create
- docker volume ls
- docker volume inspect
- docker volume rm

## [실습2-23] Volume



1. test1 volume을 생성한다.

```
$ docker volume create --name test1
```

2. volume 을 조회한다.

```
$ docker volume ls
```

3. volume을 컨테이너와 연결한다.

```
$ docker run -it -v test1:/www/test1 ubuntu:14.04  
bash
```

## [실습2-24] gogs 설치 및 실행



1. volume 생성

```
$ docker volume create --name gogs-data
```

2. gogs 컨테이너 생성

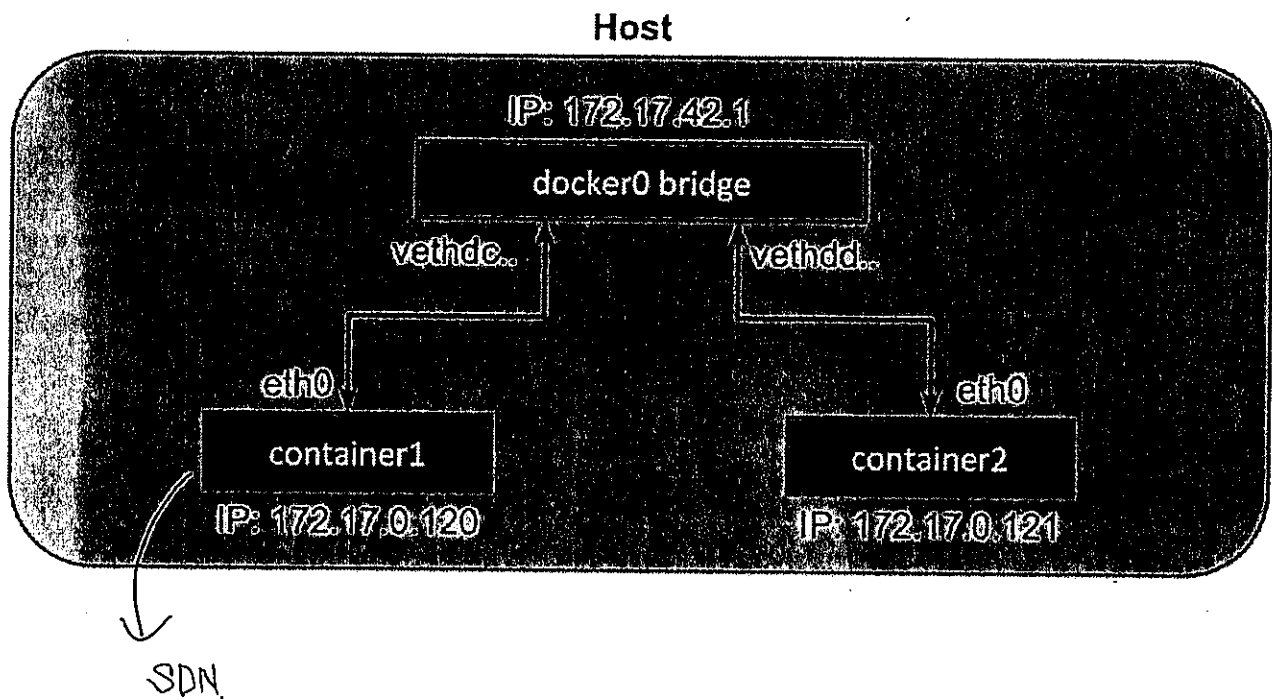
```
$ docker run -d --name=gogs -p 10022:22 -p  
3000:3000 -v gogs-data:/data gogs/gogs
```

# 컨테이너 네트워킹 Container Networking

## Networking Model

- 도커가 시작될때 호스트 머신에 docker0라고 부르는 가상 인터페이스를 생성한다.
- docker0에 사설 IP가 랜덤하게 배정된다.

# Networking Model



81

## Network 주요 명령어

- docker network create
- docker network ls
- docker network inspect
- docker network rm
- docker network connect
- docker network disconnect

82

## [실습2-25] docker0 확인



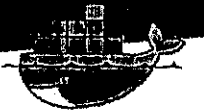
1. docker 호스트 서버로 들어간다.

```
$ docker-machine ssh default
```

2. docker0 조회

```
$ ip a
```

## [실습2-26] network 확인



1. network를 조회한다.

```
$ docker network ls
```

bridge가 docker0 이다.

2. bridge network를 검사한다.

```
$ docker network inspect bridge
```



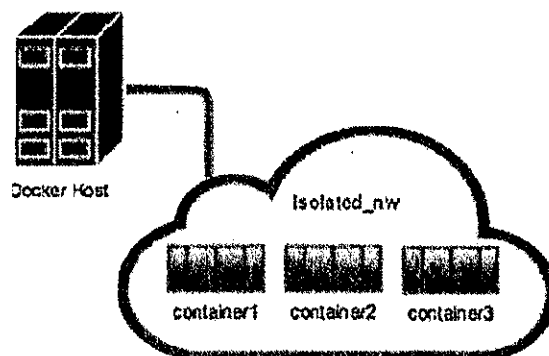
# network 유형

- bridge: docker0 같은 네트워크
- overlay: 멀티 호스트간에 연결해주는 네트워크

85

## bridge network

- 컨테이너는 동일 호스트내에 위치해야 한다.
- 사용자 정의 bridge 네트워크에 포함된 컨테이너는 컨테이너 이름으로 통신 가능



86

## [실습2-27] bridge network

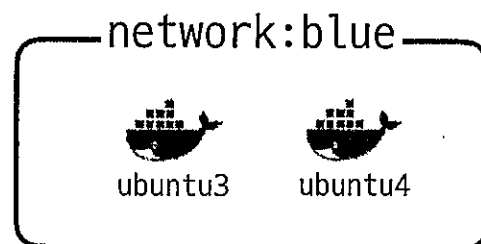
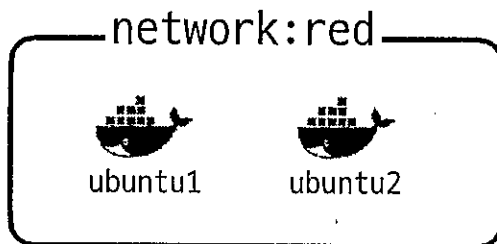


1. red, blue network를 생성한다.

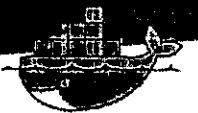
```
$ docker network create red  
$ docker network create blue
```

2. red에 ubuntu1, blue에 ubuntu3를 추가한다.

```
$ docker run -itd --net=red --name ubuntu1 ubuntu  
$ docker run -itd --net=blue --name ubuntu3 ubuntu
```



## [실습2-28] bridge network



3. red, blue network의 컨테이너 확인

```
$ docker network inspect red  
$ docker network inspect blue
```

4. red에 ubuntu2, blue에 ubuntu4를 추가한다.

```
$ docker run -itd --net=red --name ubuntu2 ubuntu  
$ docker run -it --net=blue --name ubuntu4 ubuntu
```



1. ubuntu4에서 ubuntu1,2,3 접속 확인

```
$ cat /etc/hosts  
  
$ apt-get update  
$ apt-get install inetutils-ping  
  
$ ping ubuntu2  
$ ping ubuntu3
```

2. blue와 red 네트워크 연결후 재 확인

```
$ docker network connect blue ubuntu1
```

## overlay network

### 오버레이 네트워크 요건

- key-value 스토어: Consul, Etcd, Zookeeper
- key-value 스토어에 연결된 호스트 클러스터
- 커널 버전 3.16 이상
- 각 호스트 도커 엔진 설정

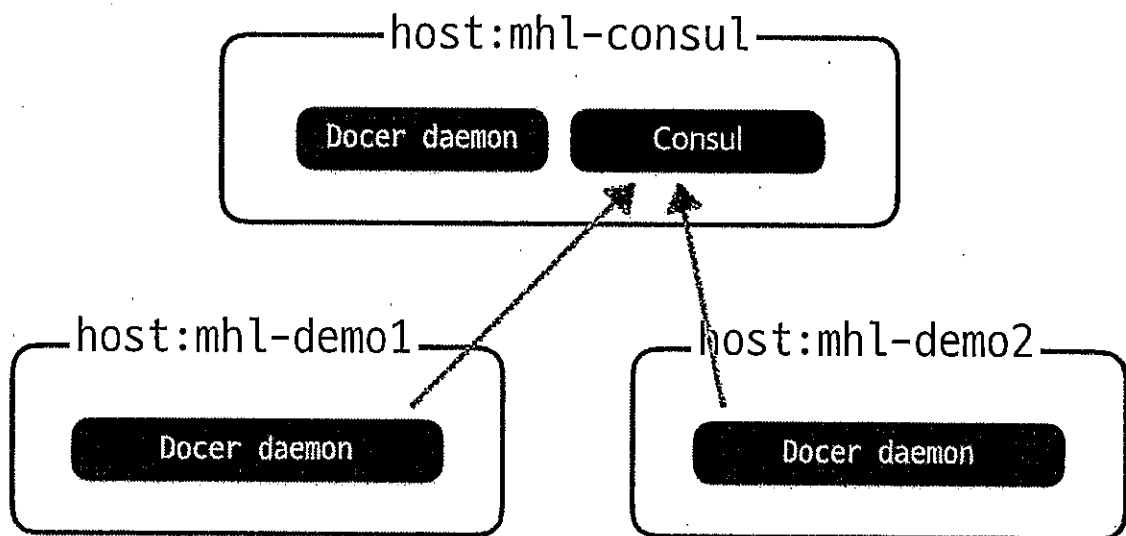
↳ 엔진과 클러스터

# overlay network

- 다른 도커 호스트에서 각각 실행중인 컨테이너들이 서로 통신할수 있게 해준다.
- 도커 엔진은 overlay 네트워크 드라이버를 통해 멀티 호스트 네트워크 지원

91

## overlay network 구성도



92

## [실습2-30] mhl-consul



### 1. mhl-consul 머신 추가

```
$ docker-machine create -d virtualbox mhl-consul
```

윈도우는 옵션추가: --virtualbox-no-vtx-check

### 2. consul 컨테이너 추가

```
$ docker $(docker-machine config mhl-consul) run -  
d -p 8500:8500 -h consul progrium/consul -server -  
bootstrap
```

## [실습2-31] mhl-demo1,demo2 추가



### 1. mhl-demo1 머신을 Consul 클러스터에 추가

```
$ docker-machine create \  
-d virtualbox \  
--engine-opt="cluster-store=consul://$(docker-  
machine ip mhl-consul):8500" \  
--engine-opt="cluster-advertise=eth1:0" \  
mhl-demo1
```

### 2. mhl-demo2 머신을 Consul 클러스터에 추가



1. mhl-demo1, 2 네트워크 조회

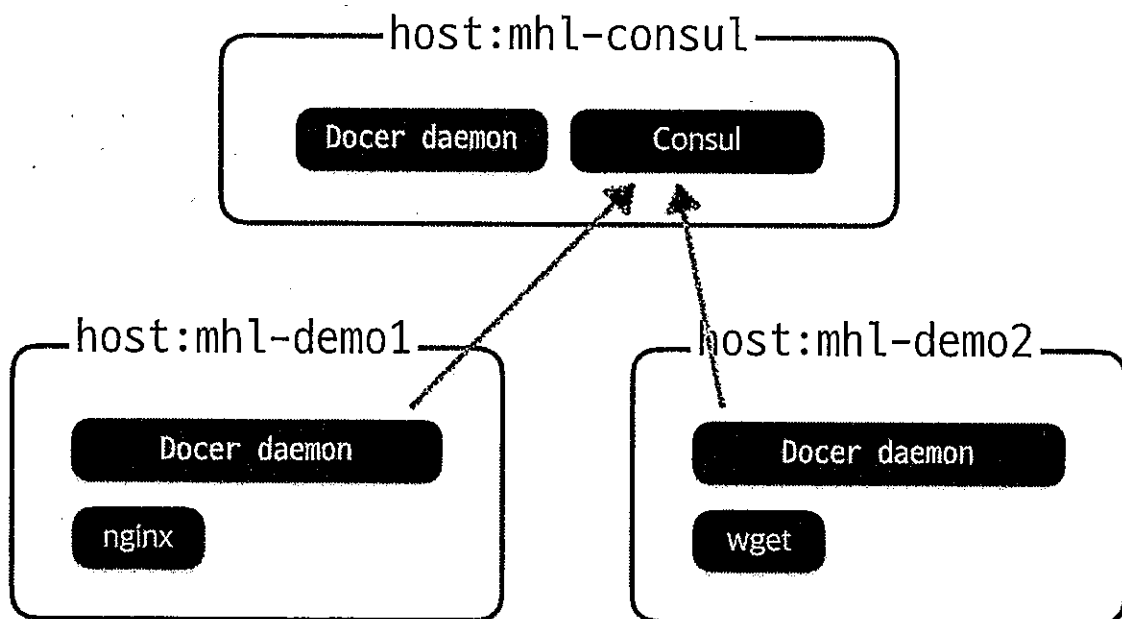
```
$ docker $(docker-machine config mhl-demo1)  
network ls
```


2. mhl-demo1에 오버레이 네트워크 frontend 추가

```
$ docker $(docker-machine config mhl-demo1)  
network create -d overlay frontend
```

3. mhl-demo2에서 frontend 조회

```
$ docker $(docker-machine config mhl-demo2)  
network ls
```





1. mhl-demo1 frontend 네트워크에 nginx 추가

```
$ docker $(docker-machine config mhl-demo1) run -  
itd --name=web --net=frontend nginx
```

2. mhl-demo2에 nginx에 접근하는 wget 실행

```
$ docker $(docker-machine config mhl-demo2) run -  
it --rm --net=frontend busybox wget -qO- http://  
web
```

## Docker Swarm



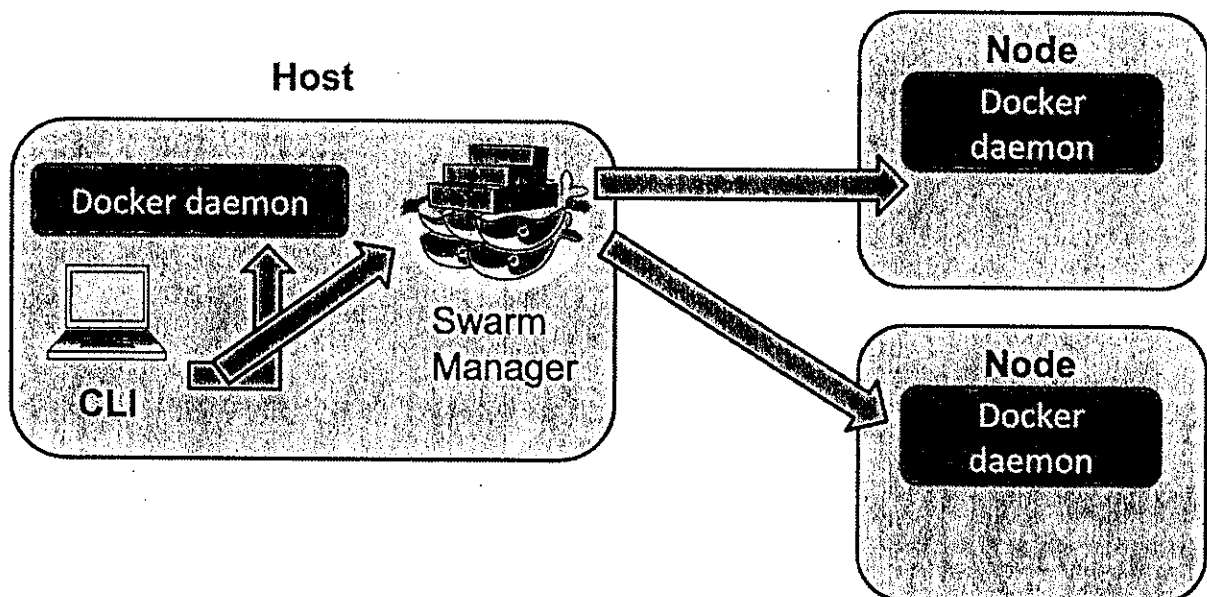
# Docker Swarm

도커 호스트 클러스터를 구성하고, 클러스터에 컨테이너를 배치해주는 도구

- 표준 Docker API
- 스케줄링 지원
- 디스커버리 지원: Consul, Etcd, ZooKeeper

99

## Swarm Manager



100



# Swarm mode

도커 엔진에 클러스터 관리가 통합된 모드

- 조건: Docker 엔진 v1.12.0 이상

특징

- 멀티 호스트 네트워킹: 분산 환경에서 여러개 노드를 하나의 네트워크로 묶은것
- 서비스 디스커버리: 멀티 호스트 환경에서 실행된 컨테이너 정보를 제공
- 로드밸런싱: 대용량 트래픽을 분산해 주는것
- 롤링 업데이트: 새로운 이미지를 순차적으로 업데이트 해 주는것

101

# Swarm mode

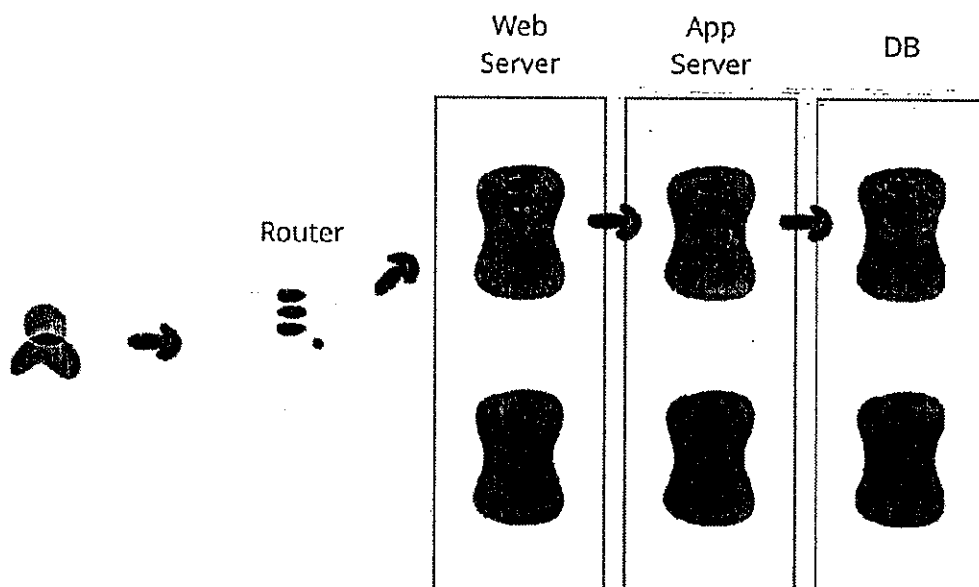
특징

- Health Check: 서비스가 정상 상태인지 확인
- 스케일 아웃: 컨테이너를 원하는 상태로 관, Desired state reconciliation
- 로깅
- 모니터링
- HA

102

# 롤링업데이트를 활용한 배포 프랙티스는?

## BlueGreen Deployment



<http://martinfowler.com/bliki/BlueGreenDeployment.html>

# Swarm mode 중요 개념

## · Swarm 클러스터

- Swarm을 이용해 구축한 클러스터
- aka Swarm

## · Node: swarm 클러스터에 참여하는 도커 엔진 인스턴스

### · Manager Node

- swarm 클러스터 상태를 관리하는 노드, 오케스트레이션
- swarm 매니저가 실행된다.
- Service 정의, Task 할당

### · Worker Node

- 매니저 노드의 명령을 받아서 컨테이너 관련 작업을 수행한다.
- Task 처리

105

# Swarm mode 중요 개념

## · Service

- 배포의 단위
- 한개 서비스는 여러개의 태스크를 갖는다.
- 보통 1개 이미지를 이용해 동일한 타입의 컨테이너를 여러개 실행한다.

### · global services

- 클러스터에 있는 모든 노드에서 실행되는 서비스

### · replicated services

- 스케일 아웃을 위해 특정 노드에서 실행하는 서비스

## · Task

- 컨테이너 배포 단위, 도커 컨테이너와 컨테이너에서 처리하는 명령
- swarm 스케줄링 단위

106

# Swarm mode 명령어

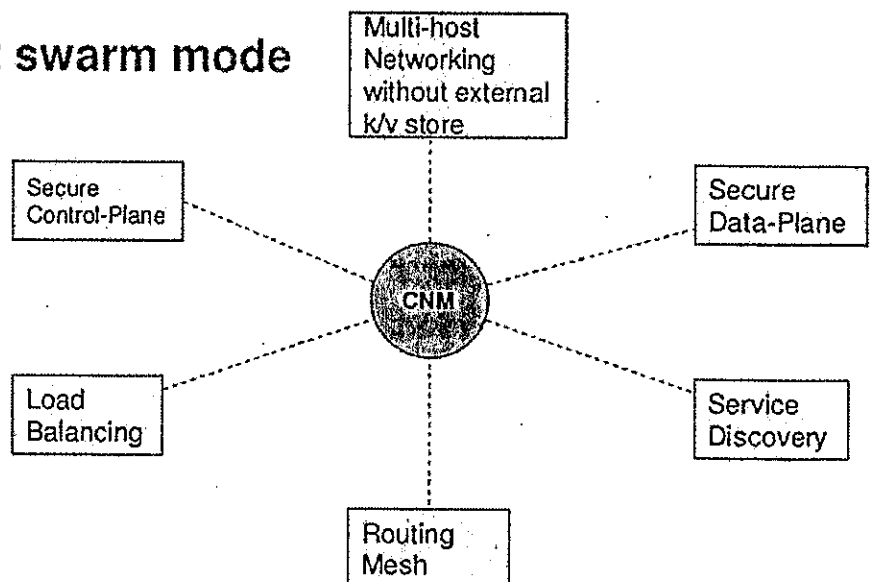
- swarm init
- swarm join
- service create
- service inspect
- service ls
- service rm
- service scale
- service ps
- service update



Docker for Ops: Docker Networking Deep Dive, Considerations and Troubleshooting by Madhu Venugopal and Jana Radhakrishnan

## New features in 1.12 swarm mode

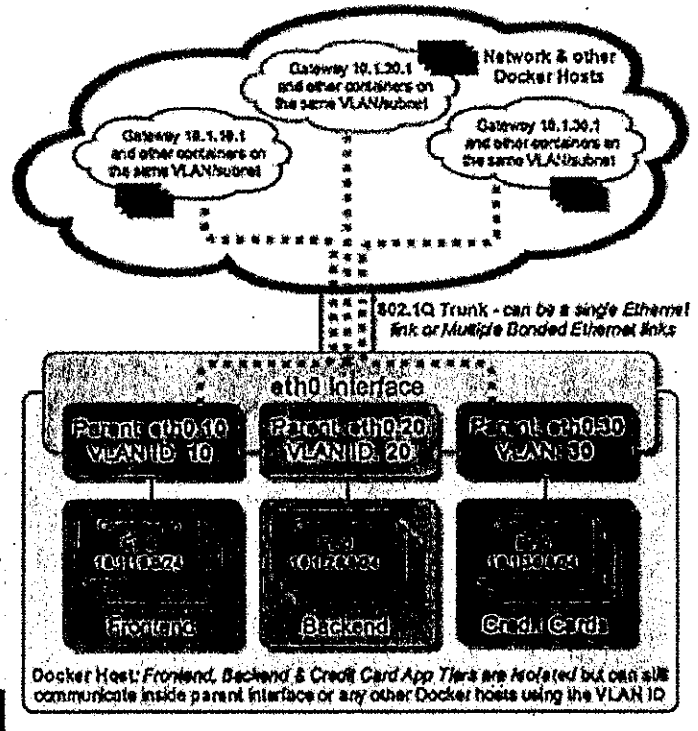
- Cluster aware
- De-centralized control plane
- Highly scalable



dockercon 16

## Macvlan driver

- Out of experimental
- Integrates with Underlay
- Place containers in your existing vlans



## [실습2-33] swl-consul

1. swl-consul 머신 추가

```
$ docker-machine create -d virtualbox swl-consul
```

2. swl-consul 머신에 consul 컨테이너 추가

```
$ docker $(docker-machine config swl-consul) run -d \
-p "8500:8500" \
-h "consul" \
progrium/consul -server -bootstrap
```



1. swarm-master를 실행할 swl-demo0 머신 추가

```
$ > $ docker-machine create \  
-d virtualbox \  
--virtualbox-disk-size 50000 \  
--swarm \  
--swarm-master \  
--swarm-discovery="consul://$(docker-machine ip  
swl-consul):8500" \  
--engine-opt="cluster-store=consul://$(docker-  
machine ip swl-consul):8500" \  
--engine-opt="cluster-advertise=eth1:0" \  
swl-demo0
```



2. swl-demo0 에 swarm-master 실행 확인

```
$ docker $(docker-machine config swl-demo0) ps
```

## [실습2-35] swl-demo1



1. swarm 클러스터에 swl-demo1 노드 추가

```
$ > $ docker-machine create \  
-d virtualbox \  
--virtualbox-disk-size 50000 \  
--swarm \  
--swarm-discovery="consul://$(docker-machine ip  
swl-consul):8500" \  
--engine-opt="cluster-store=consul://$(docker-  
machine ip swl-consul):8500" \  
--engine-opt="cluster-advertise=eth1:0" \  
swl-demo1
```

## [실습2-36] nginx 추가



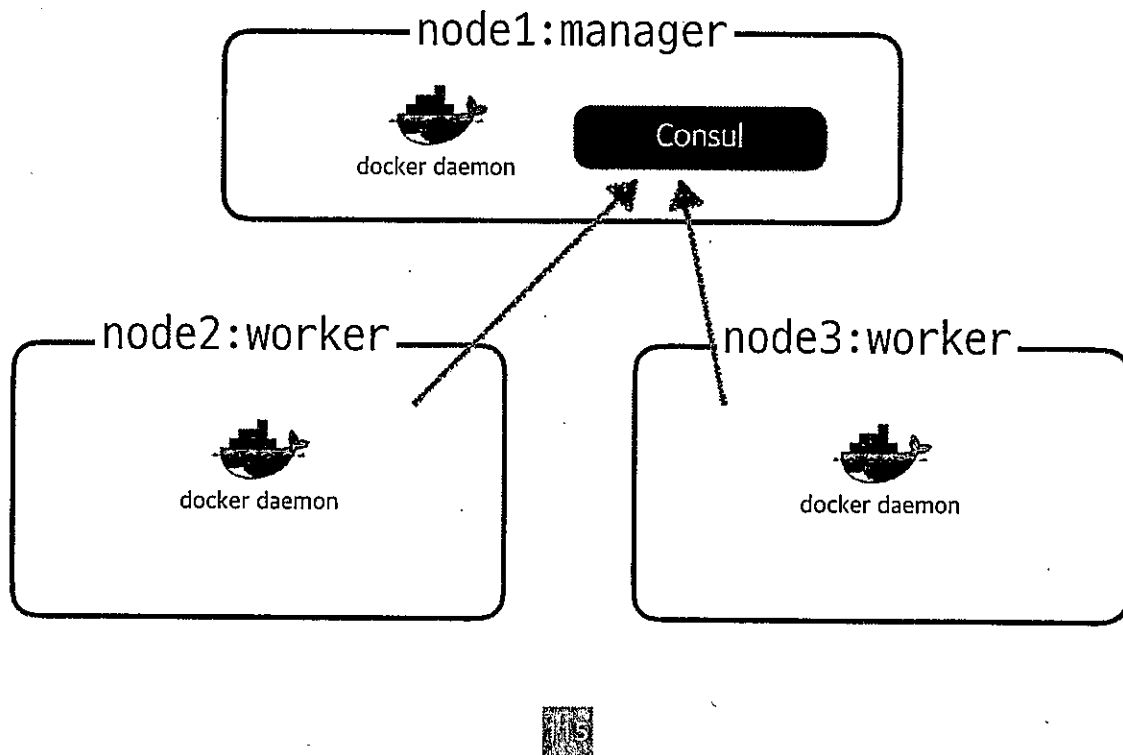
1. swarm 클러스터에 오버레이 네트워크 추가

```
$ docker $(docker-machine config swl-demo0)  
network create -d overlay frontend
```

2. swl-demo1에 nginx 추가

```
$ docker $(docker-machine config swl-demo0) run -  
itd --name=web --net= frontend --  
env="constraint:node==swl-demo1" nginx
```

# Swarm 클러스터 구성



## [실습2-37] Swarm 클러스터 생성

1. node1,2,3 를 생성한다.

```
$ docker-machine create -d virtualbox node1
```

2. 노드 확인

```
$ docker-machine ls
```

3. 매니저 노드 생성

```
$ eval $(docker-machine env node1)
$ docker swarm init --advertise-addr
192.168.99.100
```



## [실습2-38] Swarm에 워커 노드 추가



1. node2, node3를 워커 노드로 추가한다.

```
$ eval $(docker-machine env node2)
$ docker swarm join \
  --token
  SWMTKN-1-3eiiks1cs6y3qhzaqf4qcz566uh5pu9t852mhibf
  9biduqjbqd-073fekui8e4hh6kphftoi7x1m \
  192.168.99.100:2377
```

2. 워커노드 확인

```
$ docker $(docker-machine config node1) node ls
```

## [실습2-39] 서비스 추가



1. node1 에 myweb 서비스를 추가한다.

```
$ docker $(docker-machine config node1) \
  service create --name myweb -p 80:80 nginx:1.10
```

2. 서비스 확인

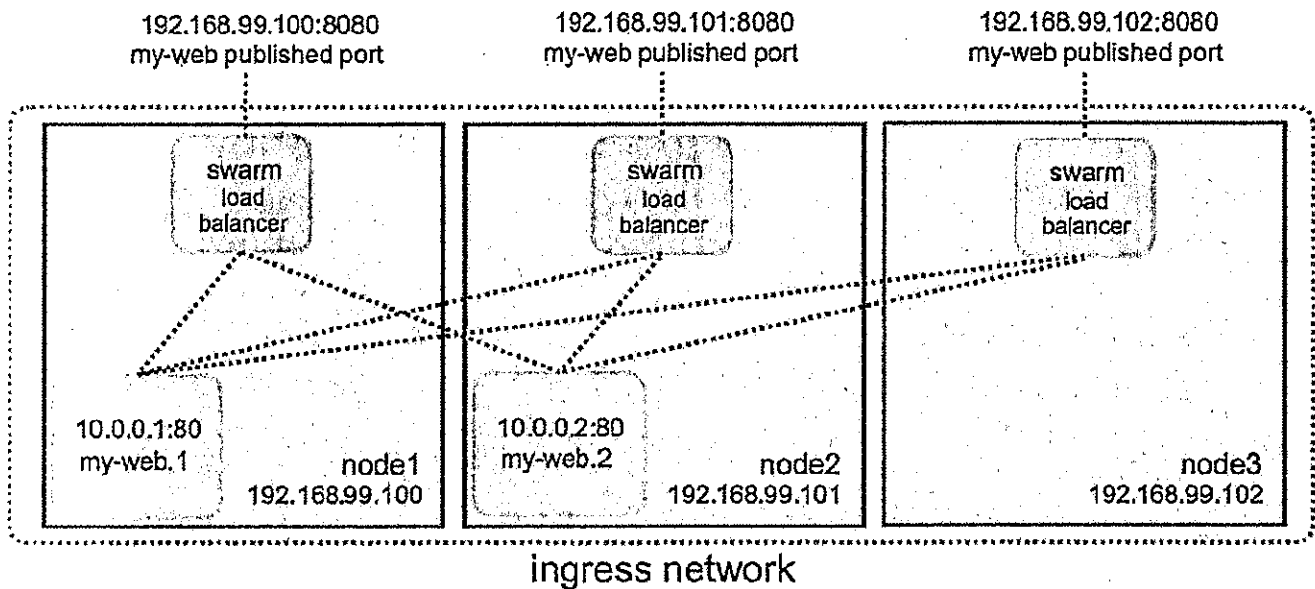
```
$ docker $(docker-machine config node1) service
ls
$ docker $(docker-machine config node1) service
ps myweb
$ curl 192.168.99.100
```

다른 노드에서도 서비스에  
접근 가능할까?

## Ingress 네트워크

- Swarm 클러스트 외부에 서비스 포트 공개를 지원
- 모든 노드는 ingress routing mesh에 들어감
- 필수 포트
  - 7946: 디스커버리 포트
  - 4789: ingress network 포트

# Ingress 네트워크



<https://docs.docker.com/engine/swarm/ingress/#publish-a-port-for-a-service>

121

## [실습2-40] 서비스 스케일아웃

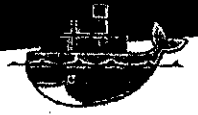


1. myweb 서비스 컨테이너를 5개로 늘린다.

```
$ docker $(docker-machine config node1) \
service scale myweb=5
```

2. 서비스 확인

```
$ docker $(docker-machine config node1) service
ps myweb
```



1. myweb 서비스 nginx 이미지를 1.11 버전으로 업데이트한다.

```
$ docker $(docker-machine config node1) \
service update \
--image nginx:1.11 myweb
```

2. 서비스 확인

```
$ docker $(docker-machine config node1) service
ls myweb
```

## SwarmKit과 Moby



# 멀티 스테이지 빌드

- 이미지 빌드의 어려움
  - 컴파일, 테스트, 패키징, etc
  - 도커 이미지 빌드전에 어플리케이션을 빌드해야 한다.
- 특징

**FROM**

**COPY --from=<stage>**

<https://ordina-jworks.github.io/conference/2017/04/18/DockerCon-Multi-Stage-Builds-And-More.html>



```
# First stage to build the application
FROM maven:3.5.0-jdk-8-alpine AS build-env
ADD ./pom.xml pom.xml
ADD ./src src/
RUN mvn clean package

# Final stage to define our minimal runtime
FROM FROM openjdk:8-jre
COPY --from=build-env target/app.jar app.jar
RUN java -jar app.jar
```

## [실습2-42] 멀티 스테이지 빌드



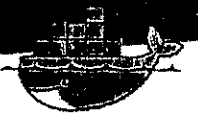
1. 웹페이지에 들어있는 <a href=""> 링크를 카운트한다.

```
FROM golang:1.7.3
WORKDIR /go/src/github.com/alexellis/href-counter/
RUN go get -d -v golang.org/x/net/html
COPY app.go .
RUN CGO_ENABLED=0 GOOS=linux go build -a -installsuffix
cgo -o app .

FROM alpine:latest
RUN apk --no-cache add ca-certificates
WORKDIR /root/
COPY --from=0 /go/src/github.com/alexellis/href-counter/
app .
CMD ["/app"]
```

Dockerfile

## [실습2-42] 멀티 스테이지 빌드



2. 이미지 빌드

```
$ docker build -t alexellis2/href-counter:latest .
```

# Docker Desktop

도커를 이용한 가상 데스크탑

- X11 (xpara)
- SSH



<https://blog.docker.com/2013/07/docker-desktop-your-desktop-over-ssh-running-inside-of-a-docker-container/>

129

## 로컬에서 클라우드까지

로컬에서 사용중인 도커 환경을 클라우드로 넘기기 위해 필요한 것들

- Images, Registry
- Build / Deploy / Run / Monitor

130

# Docker Mindset

운영체제는 상관없다.

시스템이 구동되는 머신이 무엇인지 신경쓰지 마라.

모든것은 사라진다.(ephemeral)

디스크는 읽기전용이라고 생각해라.

수정은 없다. 단지 생성하고 삭제만 있을뿐이다.

<https://www.slideshare.net/TriNimbus/from-your-desktop-to-the-cloud-things-you-need-to-consider-before-you-run-docker-in-aws>



## Swarm Kit

도커 노드 클러스터링 이슈

- 싱글 노드: private ip
- 멀티 노드





# Swarm의 진화

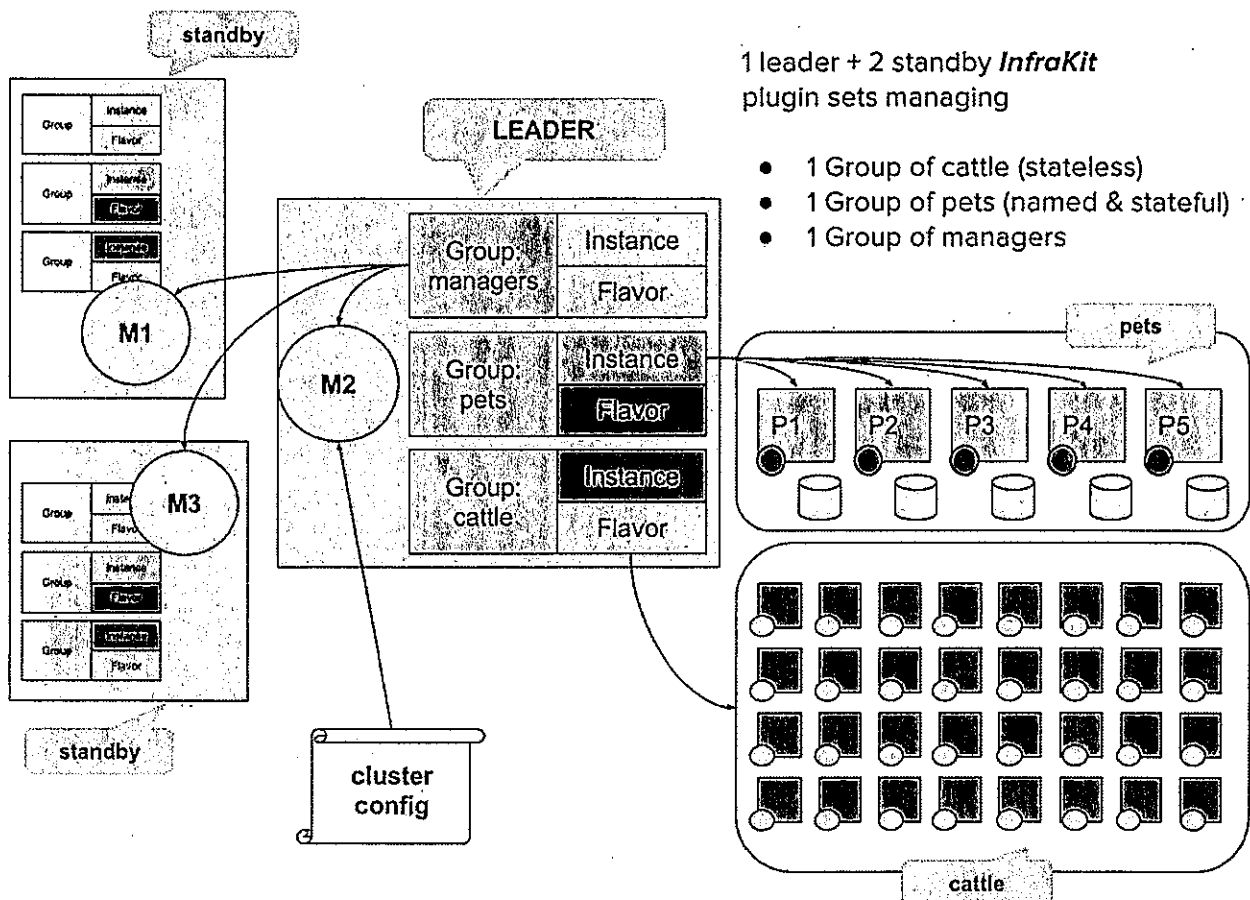
Docker Swarm

Swarm mode

InfraKit

SwarmKit

133



# SwarmKit Features

**Orchestration**

**Scheduling**

**Cluster Management**

**Security**

<https://github.com/docker/swarmkit>



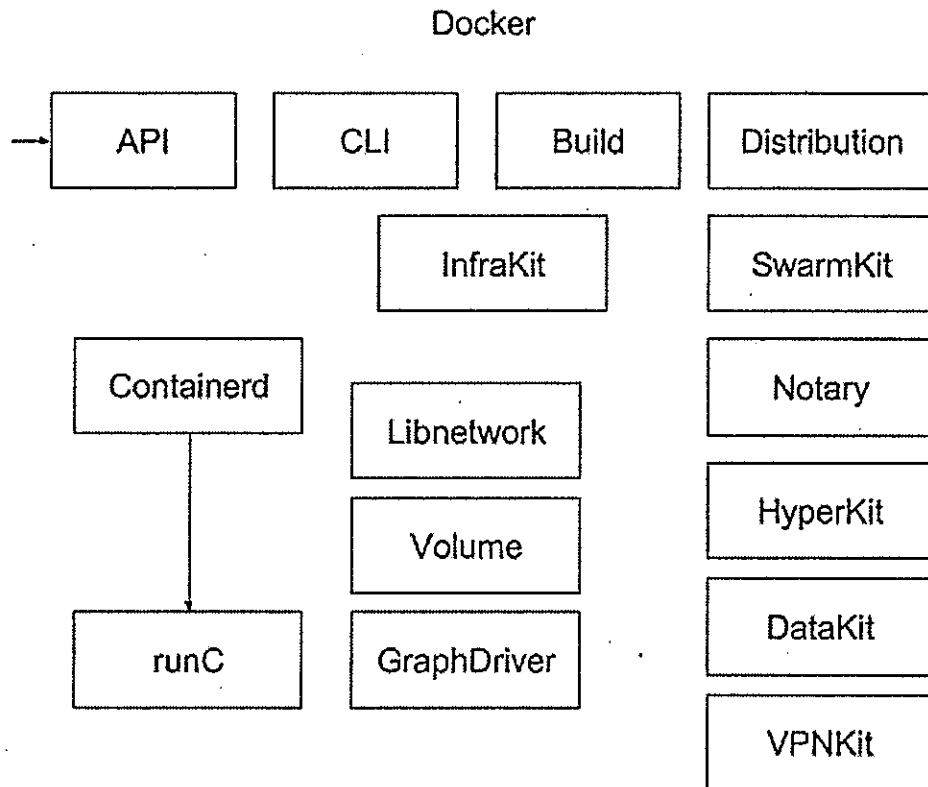
# SwarmKit Features

**\$ swarmd**

**\$ swarmctl**

<https://blog.replicated.com/engineering/first-look-at-swarmkit/>





# LinuxKit

## 도커 네이티브 실행논란

**"리눅스 컨테이너를 실행하고 싶다.  
하지만 리눅스 자체를 실행하는건 원하지 않는다."**

# LinuxKit

## 도커 실행환경

Docker for Mac, Docker for Windows

Docker for AWS, Docker for Azure, Docker for GCP



“A Toolkit for building Secure, Lean and Portable  
Linux.Subsystems”

–LinuxKit

# LinuxKit

**최소 사이즈: 35m**

**최소 부트타임**

**컨테이너 시스템 서비스**

**이뮤터블 인프라스트럭처**

<https://blog.docker.com/2017/04/introducing-linuxkit-container-os-toolkit/>



운영에 도커를 적용하는데 걸림돌

- Cloud Native Application



# Moby Project

Framework

Library

Moby Origin: reference assembly

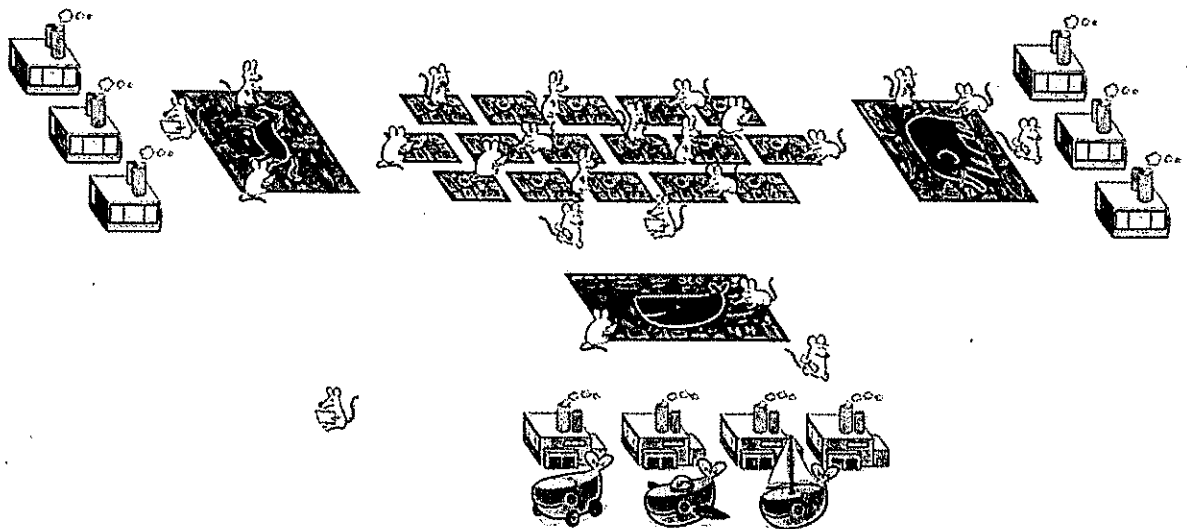
시스템 빌더

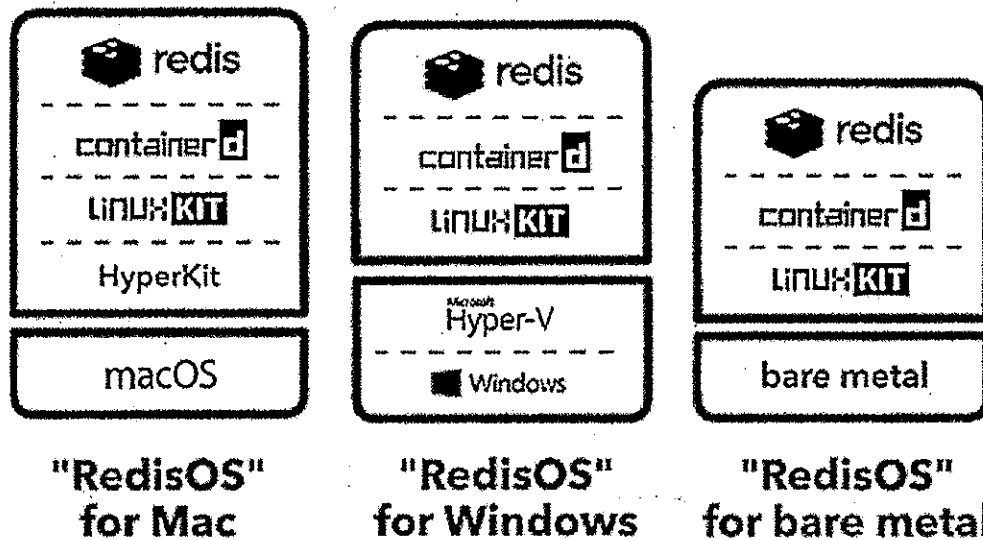


<https://blog.docker.com/2017/04/introducing-the-moby-project/>



**It's time to take our ecosystem to the next level...**  
**By collaborating on components AND COMMON ASSEMBLIES.**





[https://www.youtube.com/watch?v=hwkqiu\\_BXEO&list=PLkA60AVN3hh\\_nihZlmh6cO3n-uMdF7UIV&index=1](https://www.youtube.com/watch?v=hwkqiu_BXEO&list=PLkA60AVN3hh_nihZlmh6cO3n-uMdF7UIV&index=1)

```
kernel:
  image: "linuxkit/kernel:4.9.x"
  cmdline: "console=ttyS0 console=tty0 page_poison=1"
init:
  - linuxkit/init:cbd7ae748f0a082516501a3e914fa0c924ee941e
  - linuxkit/runc:24dfe632ed3ff53a026ee3fac046fd544434e2d6
  - linuxkit/containerd:f1130450206d4f64f0ddc13d15bb68435aa1ff61
onboot:
  - name: dhcpd
    image: "linuxkit/dhcpd:ae03169274d19fe8841314fa5a6fea3c61adb4e"
    command: ["/sbin/dhcpd", "--nobackground", "-f", "/dhcpd.conf", "-1"]
services:
  - name: redis
    image: "redis:3.0.7-alpine"
    capabilities:
      - CAP_NET_BIND_SERVICE
      - CAP_CHOWN
      - CAP_SETUID
      - CAP_SETGID
      - CAP_DAC_OVERRIDE
    net: host
outputs:
  - format: kernel+initrd
```

<https://github.com/linuxkit/linuxkit/tree/master/examples>

Q & A



espressobook

