Docker기반 DevOps 인프라 구축

3일차 / 황상철

강의 구성

Day	레벨	내용	시간	세부 내용
1 일차	기본	처음 배우는 도커	1	Docker 이해 및 환경구성
			2	Docker 이미지
			3	Docker 컨테이너
		Docker로 꾸미는 로컬 환경	4	Docker 머신
			5	컨테이너 활용
			6	Docker Compose
2 일차	활용	Docker로 꾸미는 프로젝트 환경	1	Docker Hub와 Registry
			2	개발 환경 구성
			3	도커 이미지 CI 환경구성
		Docker 멀티 호스트 구성	4	볼륨과 네트워킹
			5	Docker Swarm
			6	SwarmKit과 Moby

강의 구성

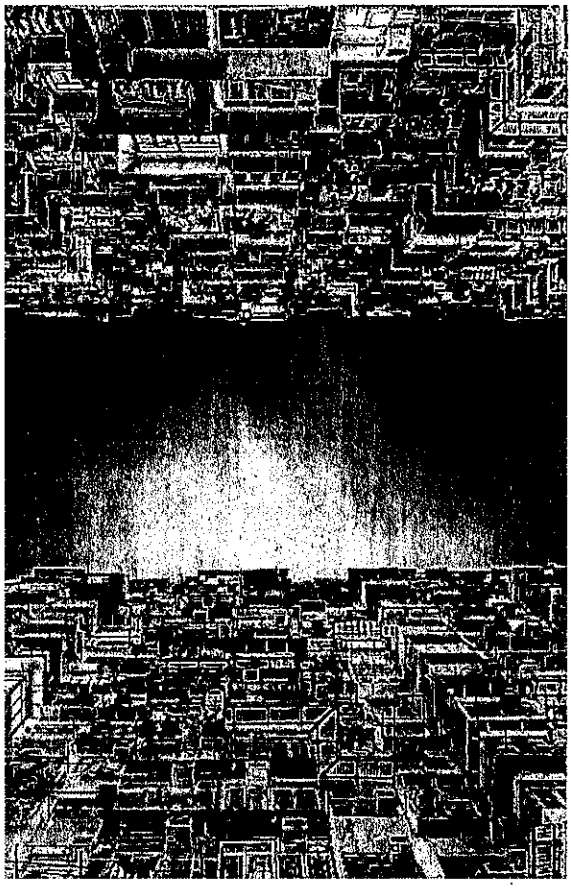
Day	레벨	내용	시간	세부 내용
3 일차	기본	Docker 운영환경	1	Docker Security
			2	Docker 모니터링
			3	Redis 클러스터
		DevOps와 도커	4	Container Linux 환경구성
			5	DevOps 환경을 위한 시스템
			6	Docker 오케스트레이션 도구

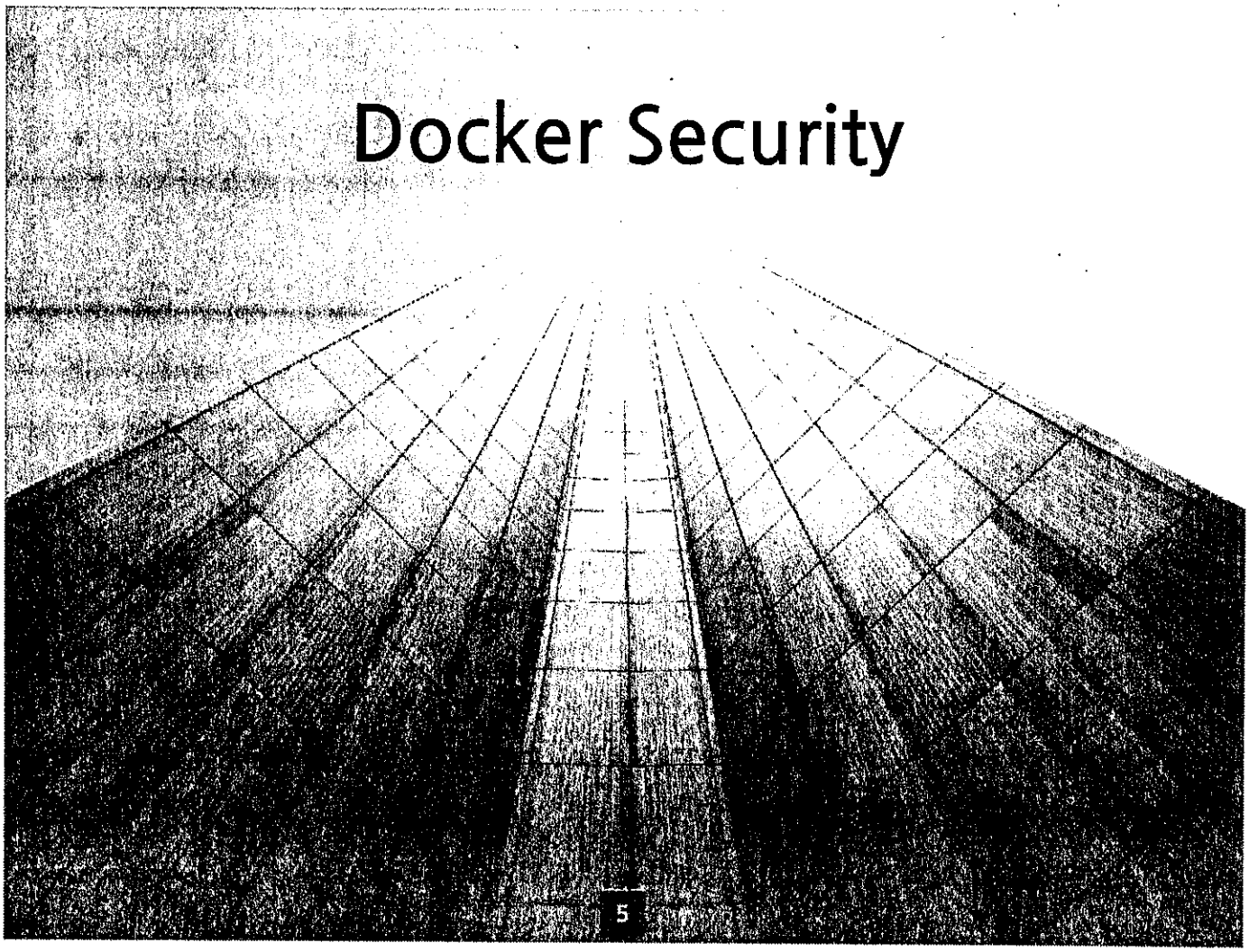
Docker 운영환경

Docker 모니터링

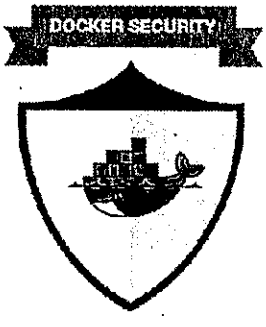
Redis 클러스터

Container Linux 환경구성





위험하다고 생각하는 이유가 잘 모르기 때문 아닐까



도커 호스트 안전

- 도커 데몬은 root 권한으로 실행된다.
- 도커 데몬은 신뢰할 수 있는 사용자만 접근가능해야 한다. ⇒ SSH 접근제어
- REST API는 TCP 소켓대신 UNIX 소켓을 사용한다.
- 도커 호스트 서버에 어드민 관리 도구를 실행하지 마라.

컨테이너가 해킹당하면

- 컨테이너에서 실행되는 프로세스는 다른 컨테이너의 프로세스에 영향을 줄 수 없다.
- 컨테이너는 자신만의 네트워크 스택을 갖는다.
  - 브릿지

https://docs.docker.com/engine/security/security/

컨테이너가 폭주하면

Docker Daemon 제어영역 내

- 컨테이너는 메모리,CPU,I/O 자원을 공유한다.
- 컨테이너는 호스트 서버 모든 자원을 소진할 수 없다.
- Kernal panic과 DoS 방지

도커 이미지 안전

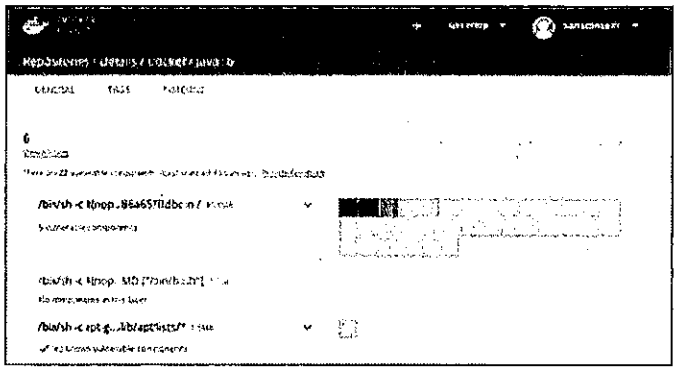
위변조

- 도커 이미지는 안전한 위치에 저장되어야 하며 체크섬을 가져야 한다.
- 배포전에 보안패치를 빌드해야 한다.
- --privileged 사용하지 않는다.

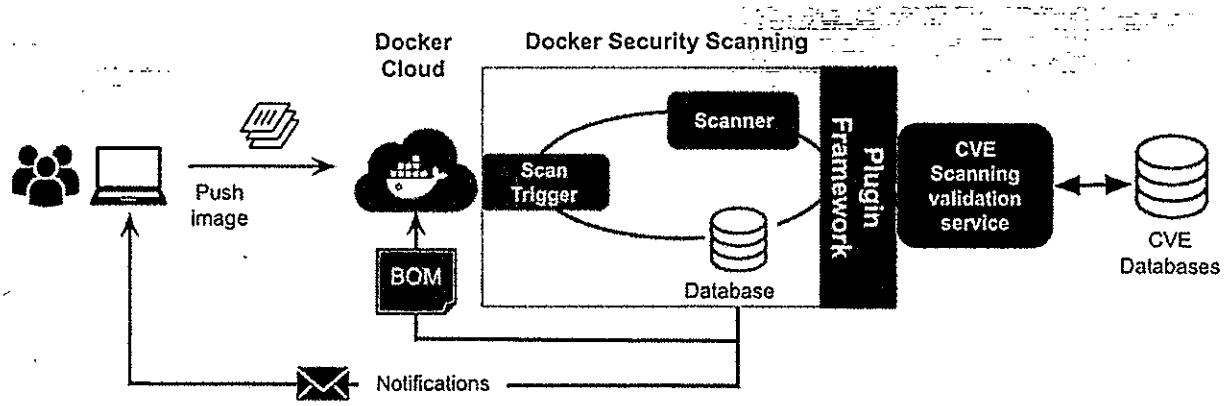
Docker Security Scanning

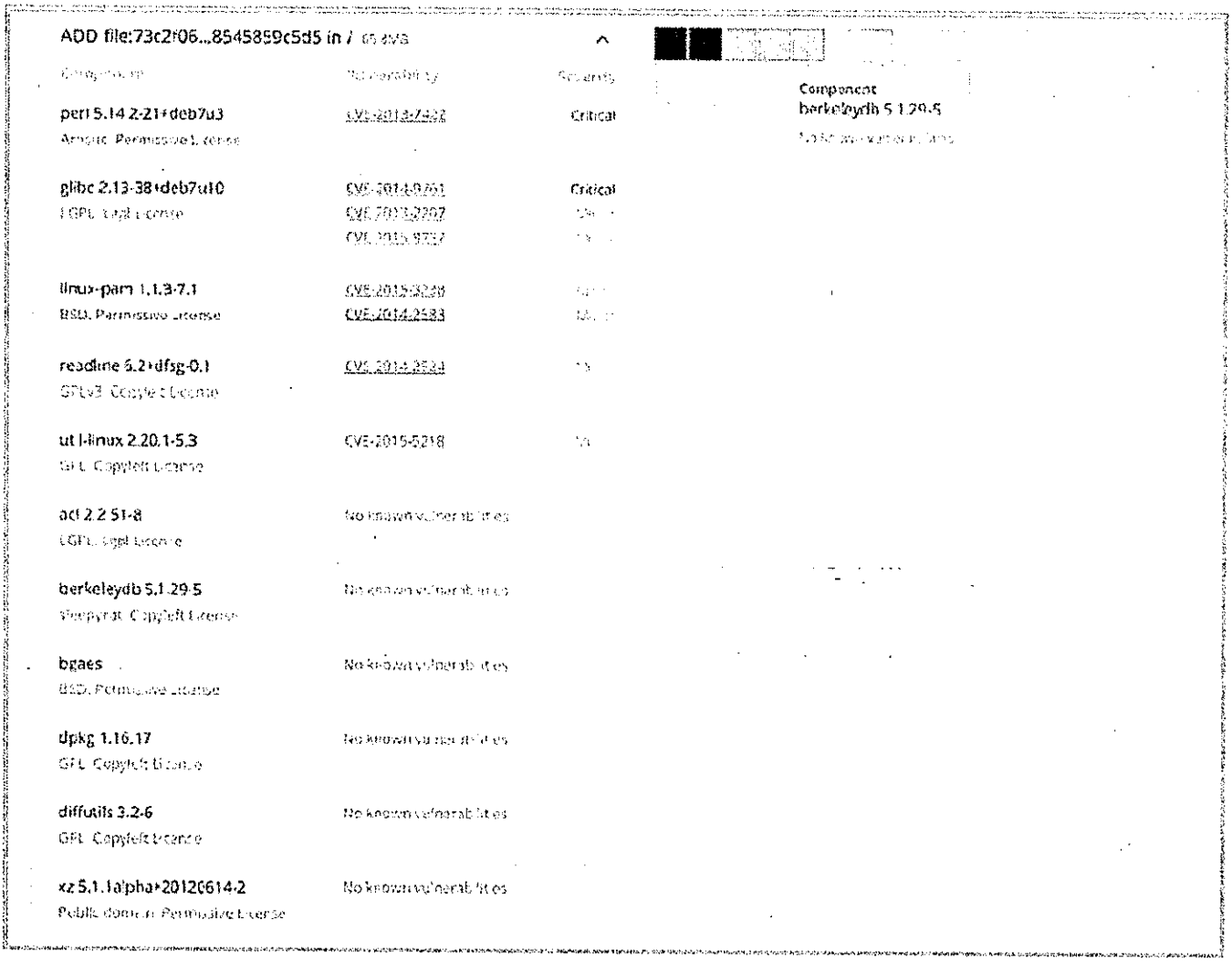
↳ Docker Hub

레지스트리에 애드온되어 이미지 보안 취약점을 스캔해서 알려준다.

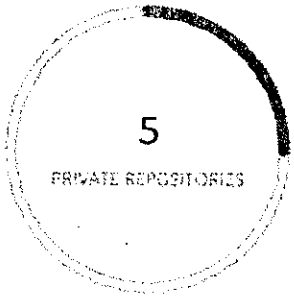


https://docs.docker.com/docker-cloud/builds/image-scan/





Plan



$ 7 /month

4 private repositories used

1 private repositories available

Change plan

Monitor with Docker Security Scan - available for your private repositories for free until August 1st, 2016

- **Natively integrated** with Docker Hub (Official and Private repos) and Docker Cloud.
- **Binary image scan** providing detection of maliciously hidden code, including stripping and renaming of components.
- **Scans are container-aware**, addresses the Docker image by layers and conducts a binary scan of the bits. Does not require running an image, and reduces the risk of undetected security problems.
- **Notifications:** Nautilus automatically notifies the user when new vulnerabilities are discovered and reported into CVE database of images which have already been scanned.
- **Comprehensive Language support:** Nautilus supports all popular programming languages, including C, C++, Java, JavaScript, Ruby, Python, C#, Go, etc.
- **Support for major OS distributions**, including Ubuntu, Debian, CentOS, etc. In addition, Docker is committed to bringing this solution to all the OSs we support in the future.
- **Displays scanning data via visualization**, provides actionable intelligence for users and allowing publishers to address newly discovered vulnerabilities.
- **Component License visibility:** Nautilus identifies the license requirements for each component within an image, allowing users to make informed choices about image selection.

Docker Bench

도커 호스트와 컨테이너에 대한 보안 취약점을 체크해주는 스크립트

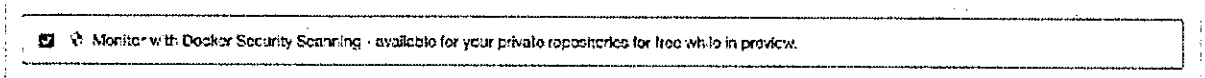
- https://github.com/docker/docker-bench-security

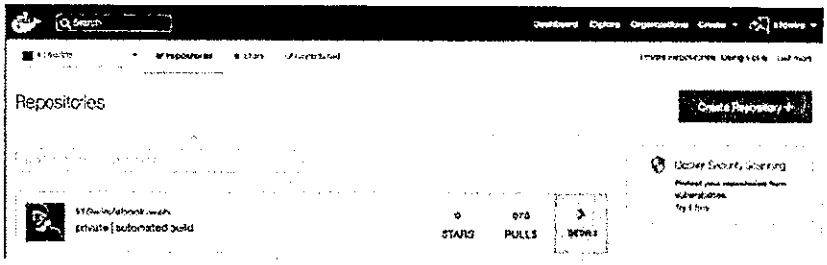
[실습2-42] Docker Security Scanning

1. docker cloud 로그인

2. 왼쪽 사이드바 Settings 선택

3. Plan 메뉴 선택





[실습2-43] Docker Bench

1. docker-bench 를 클론한다.

```
$ git clone https://github.com/docker/docker-bench-security.git
```

2.docker-compose로 실행한다.

```
$ cd docker-bench-security
$ docker-compose run --rm docker-bench-security
```

컨테이너 로깅 기준

- 로그를 이벤트 스트림으로 취급하라.
  - 로그파일로 관리하지 말고 stdout으로 출력
  - 로그파일은 앱이 아닌 실행환경으로 관리한다

https://espressobook.com/books/526

[실습3-1] 컨테이너 로그 확인

1. Docker 클라이언트로 확인

```
$ docker logs -f [컨테이너ID|컨테이너명]
```

2. Docker-compose 로 확인

```
$ docker-compose logs -f [서비스명]
```

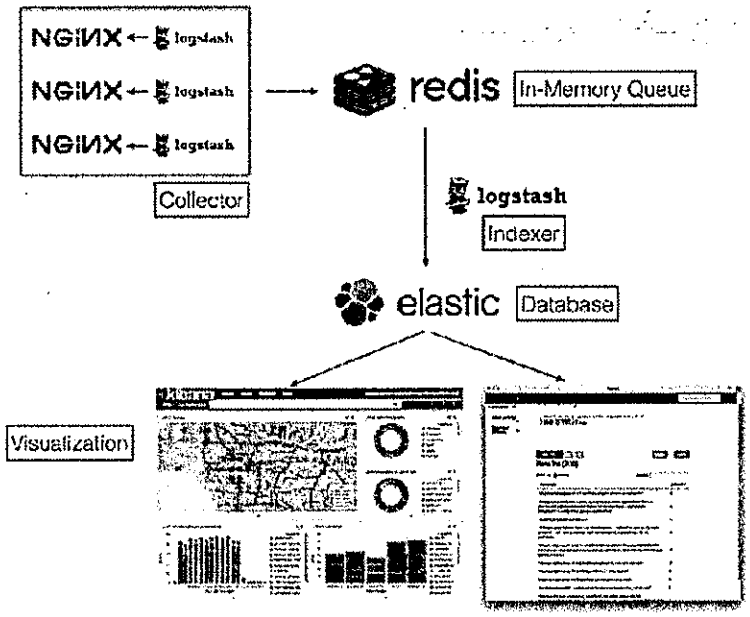
멀티호스트 환경에서 컨테이너 모니터링은?

ELK를 이용한 로그분석 환경 구축

- Elastic Search: 아파치 루신(Apache Lucene) 기반의 검색엔진 서버
- Logstash: 로그와 같은 데이터를 실시간으로 수집해주는 엔진
- Kibina: 엘라스틱 서치 결과를 분석하고 표현해주는 플랫폼
- (Redis): 인메모리 데이터 저장소

⇒ Queuing

ELK를 이용한 로그분석 환경 구축



[실습3-2] 싱글노드 ELK ElasticSearch

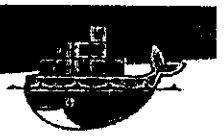
1. 싱글노드 머신을 추가한다.

최소 메모리

```
$ docker-machine create -d virtualbox --
virtualbox-memory "2048" elkr

$ docker-machine ssh elkr

$ sudo sysctl -w vm.max_map_count=262144
```

파일개수

[실습3-2] 싱글노드 ELK ElasticSearch



2. ElasticSearch 컨테이너를 추가한다.

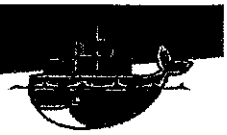
```
$ docker $(docker-machine config elkr) run -d \
 -p 9200:9200 --name elasticsearch
docker.elastic.co/elasticsearch/elasticsearch:
5.3.0
```

3. ElasticSearch 컨테이너 확인 (기본유저 elastic/chagneme)

```
$ curl http://$(docker-machine ip elkr):9200
```

https://www.elastic.co/guide/en/elasticsearch/reference/current/docker.html

[실습3-2] 싱글노드 ELK ElasticSearch



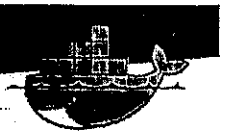
4. ElasticHQ 플러그인을 설치한다.

```
$ docker-machine ssh elkr

$ docker exec -it elasticsearch /bin/bash

> root# /usr/share/elasticsearch/bin/
elasticsearch-plugin install royrusso/
elasticsearch-HQ/v1.0.0
```

5. ElasticHQ 플러그인 확인

```
$ curl http://$(docker-machine ip elkr):9200/
_plugin/hq/
```

[실습3-2] 싱글노드 ELK ElasticSearch



6. ElasticSQL 플러그인을 설치한다.

```
> root# /usr/share/elasticsearch/bin/plugin
install https://github.com/NLPchina/
elasticsearch-sql/releases/download/2.1.0/
elasticsearch-sql-2.1.0.zip
```

7. ElasticSQL 플러그인을 확인한다.

```
$ curl http://$(docker-machine ip elkr):9200/
_plugin/sql/
```

[실습3-3] 싱글노드 ELK Kibana

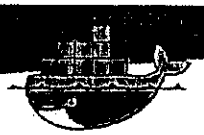
1. kibana 컨테이너를 추가한다.

```
$ docker $(docker-machine config elkr) \
 run -d --link elasticsearch:elasticsearch \
 --name kibana -p 5601:5601 kibana
```

2. kibana 컨테이너 확인

```
$ curl http://$(docker-machine ip elkr):5601
```

[실습3-4] 싱글노드 ELK Redis



1. redis 컨테이너를 추가한다.

```
$ docker $(docker-machine config elkr) \
run -d -p 6379:6379 --name redis redis
```

2. redis 컨테이너 확인

```
$ docker-machine ssh elkr

elkr0$ docker exec -it redis redis-cli info
```

[실습3-5] 싱글노드 ELK Web1

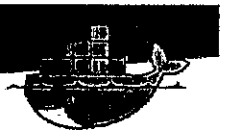
1. web 노드를 추가한다.

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

2. json 포맷 로그를 남기는 nginx.conf 파일

```
log_format json_format '{"@time": "$time_iso8601", '
      '"@fields": { '
      '"host": "$http_host", '
      '"ip": "$remote_addr", '
      '"request-time": "$request_time", '
      '"status": "$status", '
      '"request": "$request", '
      '"size": "$body_bytes_sent", '
      '"user-agent": "$http_user_agent", '
      '"referrer": "$http_referer" } }';

access_log  /var/log/nginx/access.log json_format;
```

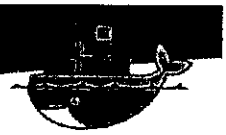
[실습3-5] 싱글노드 ELK Web1



3. web 노드에 nginx.conf,index.html 파일을 복사한다.

```
$ docker-machine scp ./nginx.conf web:~/
$ docker-machine scp ./index.html web:~/

$ docker-machine ssh web

$ cp index.html ~/nginx/content
```

[실습3-5] 싱글노드 ELK Web1



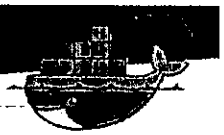
4. web 노드에 web1 으로 nginx를 추가한다.

```
$ docker run -d -P --name web1 \
 -v ~/nginx.conf:/etc/nginx/nginx.conf \
 -v ~/nginx/log:/var/log/nginx \
 -v ~/nginx/content:/usr/share/nginx/html \
 nginx
```

5. 로그를 확인한다.

```
$ tail -f /home/docker/nginx/log/access.log
```

[실습3-6] 싱글노드 ELK Logstash



1. nginx 로그를 redis로 유입시키는 log-collector.conf 정의

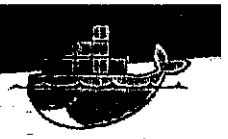
```
input {
  file {
    path => "/nginx/log/access.log"
    type => nginx
    codec => json
  }
}
filter {
  geoip {
    source => "[@fields][ip]"
  }
}
output {
  redis {
    host => "redis"
    port => "6379"
    data_type => "list"
    key => "logstash"
  }
}
```

[실습3-6] 싱글노드 ELK Logstash

2. nginx 로그를 redis로 집어넣는 Logstash 컨테이너 설치

```
$ docker run -it --rm \
 --link redis:redis \
 -v /home/docker/nginx/log:/nginx/log \
 -v ~/logstash:/config-dir \
 logstash logstash -f /config-dir/log-
collector.conf
```

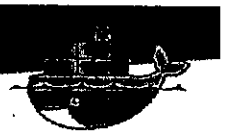
[실습3-7] 싱글노드 ELK Logstash



3. redis 로그를 Elastic Search로 인덱싱하는 Logstash 컨테이너 설치

```
$ docker run -it --rm \
 --link redis:redis \
 --link elasticsearch:elasticsearch \
 -v ~/logstash:/config-dir \
 logstash logstash -f /config-dir/log-
indexer.conf
```

[실습3-8] 멀티노드 ELK 네트워크



1. elkr 네트워크를 추가한다.

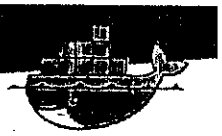
```
$ docker $(docker-machine config mhl-elkr)
network create -d overlay elkr
```

2. web 네트워크를 추가한다.

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

web

[실습3-9] 멀티노드 ELK ElasticSearch



어떤 네트워크에 추가?

1. elasticsearch 컨테이너를 elkr 네트워크에 추가한다.

```
$ docker $(docker-machine config mhl-elkr) \
 run -d -p 9200:9200 \
 --net=elkr --name elasticsearch \
 elasticsearch
```

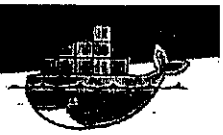
2. ElasticHQ, ElasticSQL 플러그인을 설치한다.

[실습3-10] 멀티노드 ELK Kibana

1. elasticsearch 컨테이너를 elkr 네트워크에 추가한다.

```
$ docker $(docker-machine config mhl-elkr) \
 run -d --net=elkr  \
-e ELASTICSEARCH_URL=http://elasticsearch:9200 \
--name kibana -p 5601:5601 kibana
```

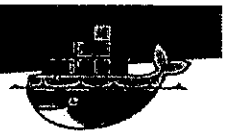
[실습3-11] 멀티노드 ELK Redis



1. redis 컨테이너를 elkr 네트워크에 추가한다.

```
$ docker $(docker-machine config mhl-elkr) \
 run -d --net=elkr  \
 --name redis -p 6379:6379  \
 redis
```

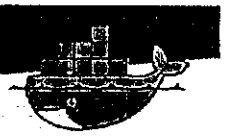
[실습3-12] 멀티노드 ELK Logstash



1. redis 로그를 elasticsearch 로 인덱싱하는 Logstash 컨테이너를 elkr 네트워크에 추가한다.

```
$ docker run -it --rm \
 --net=elkr  \
 -v ~/logstash:/config-dir \
logstash logstash -f /config-dir/log-indexer.conf
```

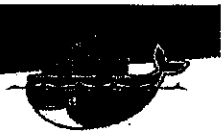
[실습3-13] 멀티노드 ELK Web



1. mhl-web 노드 web 네트워크에 nginx 컨테이너 추가

```
$ docker run -d -P --name web1 \
 --net=web  \
 -v ~/nginx.conf:/etc/nginx/nginx.conf \
 -v ~/nginx/log:/var/log/nginx \
 -v ~/nginx/content:/usr/share/nginx/html \
 nginx
```

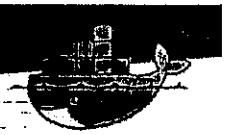
[실습3-13] 멀티노드 ELK Web



2. nginx 로그를 redis 로 유입시키는 Logstash 컨테이너 추가

```
$ docker run -it --rm \
 --net=elkr  \
 -v /home/docker/nginx/log:/nginx/log \
 -v ~/logstash:/config-dir \
 logstash logstash -f /config-dir/log-
collector.conf
```

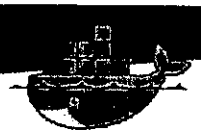
[실습3-14] docker-compose로 ELK 구축



1. elkr 노드 추가

```
$ docker-machine create --driver virtualbox \
 --virtualbox-disk-size 10000 --virtualbox-memory
2048 elkr

$ docker-machine ssh elkr

$ sudo sysctl -w vm.max_map_count=262144
```

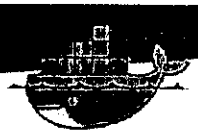
[실습3-14] docker-compose로 ELK 구축



2. elkr 노드에 docker-compose 설치

```
$ docker-machine ssh elkr
sudo -i

curl -L "https://github.com/docker/compose/
releases/download/1.11.2/docker-compose-$(uname -
s)-$(uname -m)" -o /usr/local/bin/docker-compose

chmod +x /usr/local/bin/docker-compose
```

[실습3-14] docker-compose로 ELK 구축



3. 컨테이너 설정파일 elkr 노드에 복사

```
$ git clone https://github.com/k16wire/elk.git

$ cd elk
```

- elkr.yml : elkr 컨테이너를 위한 yml
- nginx-logcollector.yml: nginx, logstash 컨테이너를 위한 yml

[실습3-14] docker-compose로 ELK 구축

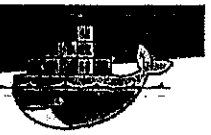
elkr.yml

```
elasticsearch:
  image: elasticsearch
  ports:
    - 9200:9200
kibana:
  image: kibana
  ports:
   - 5601:5601
  links:
   - elasticsearch:elasticsearch
redis:
  image: redis
  ports:
   - 6379:6379
```

nginx-logcollector.yml

```
nginx:
  image: nginx
  ports:
    - 80
  volumes:
   - /home/docker/nginx.conf:/etc/nginx/
nginx.conf
   - /home/docker/nginx/log:/var/log/nginx
   - /home/docker/nginx/content:/usr/share/nginx/
html
logcollector:
  image: logstash
  external_links:
    - elk_redis_1:redis
  volumes:
    - /home/docker/nginx/log:/nginx/log
    - /home/docker/logstash:/config-dir
  command: logstash logstash -f /config-dir/log-
collector.conf
```

[실습3-14] docker-compose로 ELK 구축



4. elasticsearch, kibana, logstash, nginx 컨테이너 실행

```
$ docker-compose -f elkr.yml up -d
$ docker-compose -f nginx-logcollector.yml up -d
$ docker-compose -f log-indexer.yml up -d
```

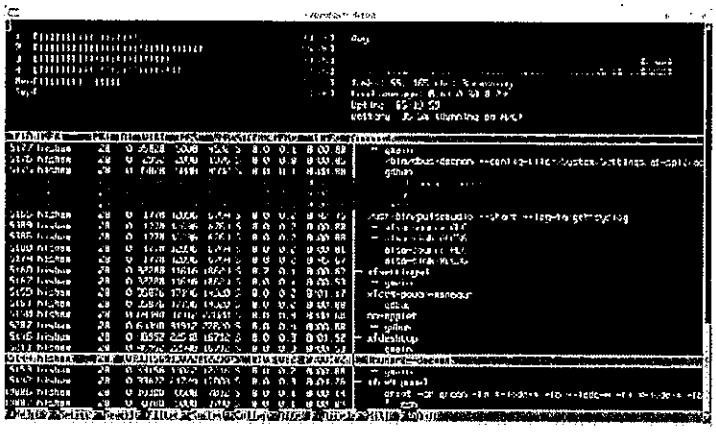
도커는 격리된 컨테이너 내부에서 프로세스를 실행한다.

컨테이너는 호스트에서 실행되는 프로세스이다.

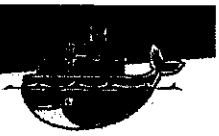
https://docs.docker.com/engine/reference/run/

htop

- 터미널 기반 유닉스 프로세스 모니터링 도구
- http://hisham.hm/htop/



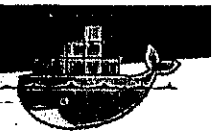
[실습3-15] htop 으로 호스트 모니터링



1. 호스트를 모니터링하는 htop 컨테이너 실행

```
$ docker run -it --rm --pid host tehbilly/htop
```

[실습3-15] htop으로 컨테이너 모니터링



1. nginx 컨테이너 실행

```
$ docker run -d -P nginx
```

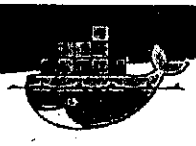
2. htop으로 myweb 컨테이너 모니터링

```
$ docker run -it --rm \
 --pid=container:myweb tehbilly/htop
```

Sysdig

- 통합 시스템 관리 도구
- http://www.sysdig.org/

[실습3-16] sysdig으로 호스트 모니터링



1. sysdig 컨테이너 실행

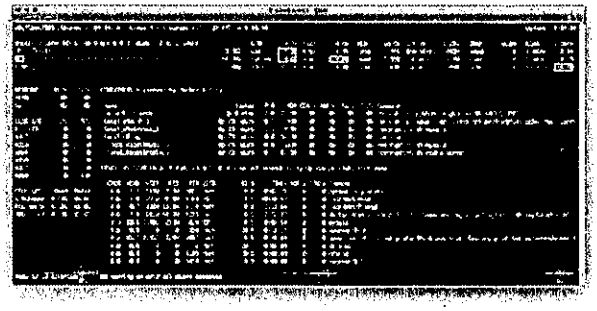
```
$ docker run -it --rm --name=sysdig --
privileged=true \
 --volume=/var/run/docker.sock:/host/var/run/
docker.sock \
 --volume=/dev:/host/dev \
 --volume=/proc:/host/proc:ro \
 --volume=/boot:/host/boot:ro \
 --volume=/lib/modules:/host/lib/modules:ro \
 --volume=/usr:/host/usr:ro \
 sysdig/sysdig
```

2. csysdig 도구 실행

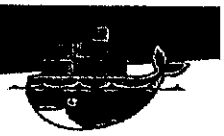
```
$ csysdig
```

Glances

- 파이썬으로 만든 시스템 모니터링 도구
- CLI,API,Web 인터페이스 지원
- https://nicolargo.github.io/glances/



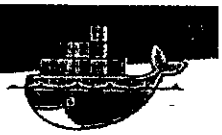
[실습3-17] glances 콘솔



1. glances 컨테이너 실행

```
$ docker run -it --pid=host \
 -v /var/run/docker.sock:/var/run/docker.sock:ro \
 --name glances \
 nicolargo/glances
```

[실습3-18] glances 웹콘솔



1. glances 웹 콘솔 컨테이너 실행

```
$ docker run -d -p 61208:61208 -p 61209:61209 -e
"GLANCES_OPT=-w" --pid=host -v /var/run/
docker.sock:/var/run/docker.sock:ro --name glances
nicolargo/glances
```

2. 브라우저로 웹 콘솔 접속

http://192.168.99.100:61208/10

[실습3-18] glances 웹콘솔

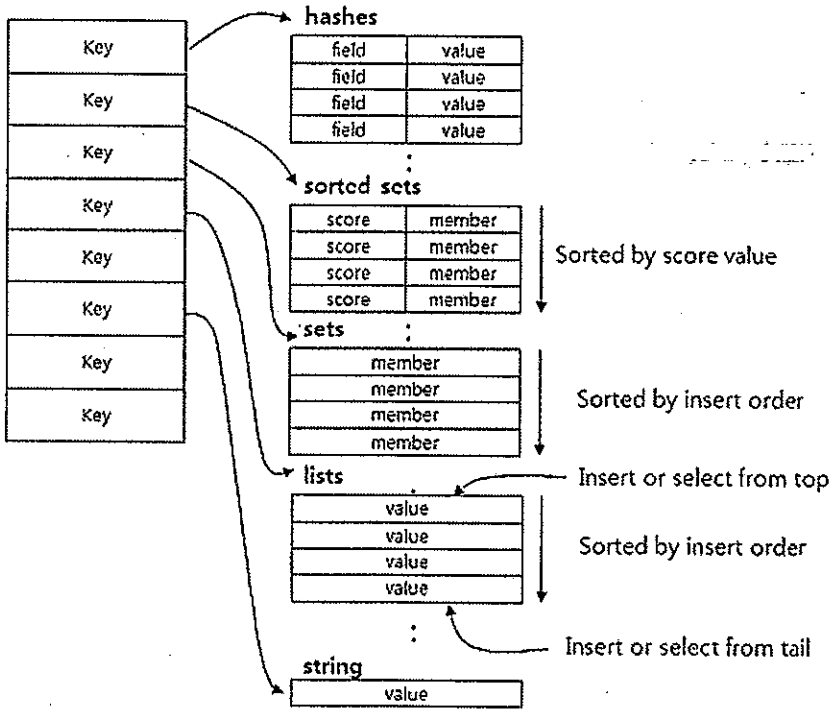
3. REST API

http://192.168.99.100:61208/api/2/pluginslist

redis

- REmote DIctionary System
- 메모리 기반의 Key/Value 스토어
- 저장 가능한 데이터 용량은 물리적인 메모리 크기를 넘어설 수 없다.
- 서버 재시작을 위해 Disk에 데이터를 저장

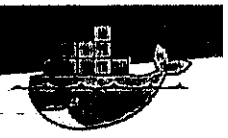
redis 데이터 구조



redis-cli

- set key "value"
- get key
- mset a apple b bird
- mget a b
- del a
- ping
- exist a

[실습3-19] redis 서버 실행



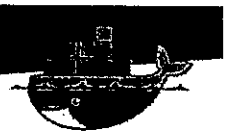
1. redis 컨테이너 실행

```
$ docker run -d -p 6379:6379 --name fc-redis redis
```

2. 스토리지 저장 옵션으로 실행

```
$ docker run -d --name fc-redis \
-v /apps/redis:/data \
redis redis-server --appendonly yes
```

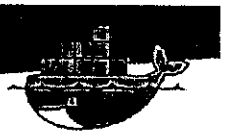
[실습3-20] redis-cli 확인



1. redis-cli 컨테이너 실행

```
$ docker run -it --link fcredis:redis --rm redis
redis-cli -h redis -p 6379
```

[실습3-21] redis 이미지 빌드



1. redis.conf 파일을 작성한다.

```
port 6379
timeout 300
databases 1
requirepass password1
maxclients 1000
appendonly no
tcp-keepalive 60
syslog-enabled yes
```

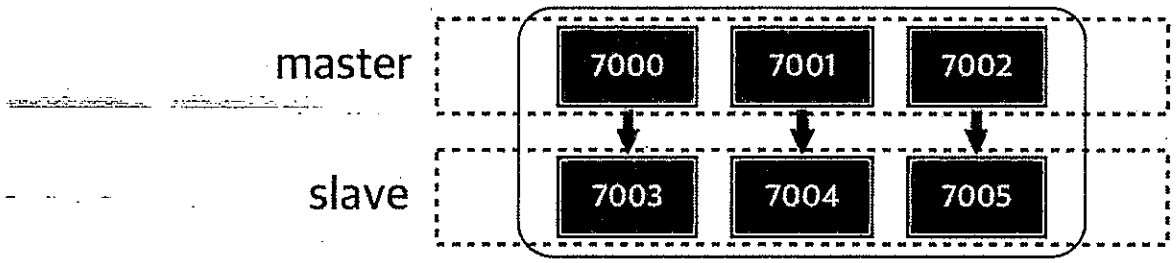
2. Dockerfile 작성한다.

```
FROM redis:3.0

COPY redis.conf /usr/local/etc/redis/redis.conf
CMD [ "redis-server", "/usr/local/etc/redis/redis.conf" ]
```

[실습3-22] redis-cluster 구성

1. redis 클러스터 도커 컨테이너 실행

```
$ docker run -d -P grokzen/redis-cluster:3.0.6
```



https://hub.docker.com/r/grokzen/redis-cluster/

http://rocksea.tistory.com/339

Vagrant

- 다양한 가상머신을 동일한 방식으로 만들고 관리할 수 있게 해주는 도구
- VirtualBox, VMware, AWS 등 지원
- 루비 언어를 따르는 Vagrantfile 에 필요한 설정을 정의
- https://www.vagrantup.com/

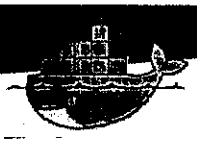
Vagrant Box

- Vagrant 로 만들 수 있는 가상서버 유형
- Box 리스트: https://atlas.hashicorp.com/boxes/search

Vagrantfile에 포함되는 내용

- Box: 패키징할 서버 유형
- 포트 바인딩
- 네트워크 주소
- 이름, CPU, Memory
- 부트 스트래핑: 어플리케이션 설치
- 프로비저닝: 어플리케이션 설정
- 폴더 마운트

[실습3-23] Vagrant 설치



맥,윈도우 환경에 맞는 Vagrant 파일을 설치한다.

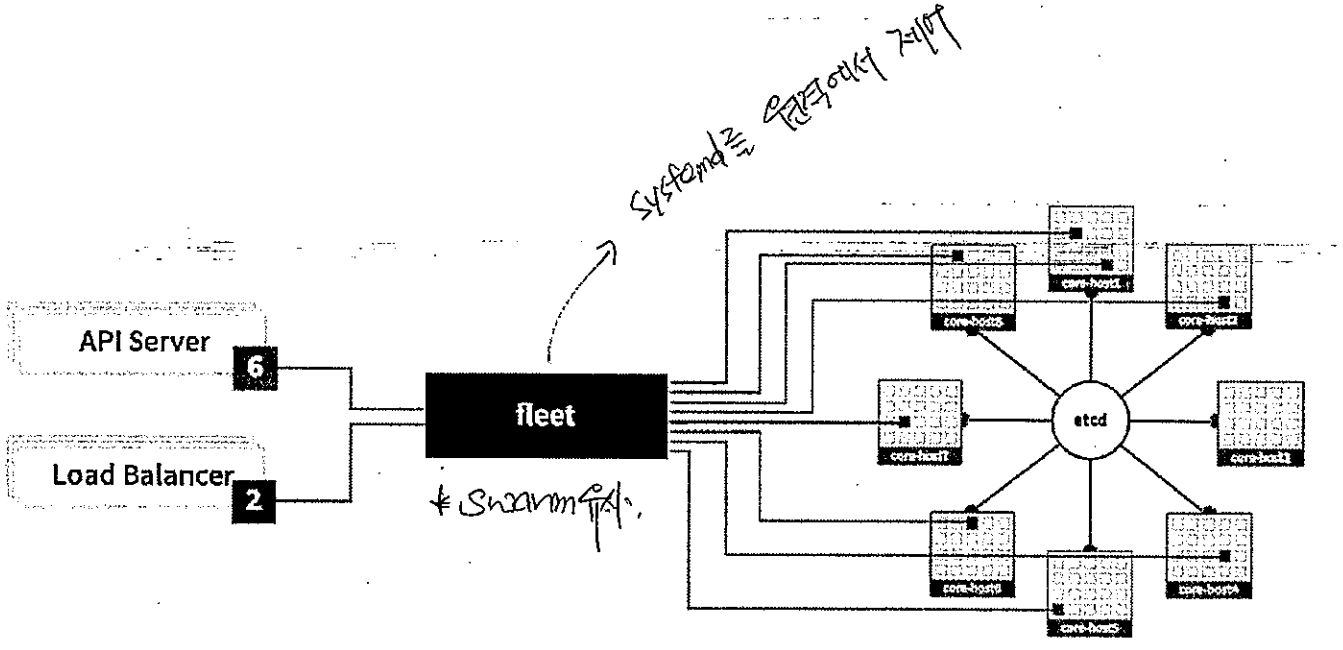
https://www.vagrantup.com/downloads.html

CoreOS

이전 → Container OS

Docker 호스트에 특화된 리눅스

- 최소화된 메모리 사용(40%)
- systemd, etcd, fleet
- CoreUpdate(상용)
- https://coreos.com/



[실습3-24] CoreOS 단일 머신

1. CoreOS 설치를 위한 Vagratntfile 을 받는다.

```
$ git clone https://github.com/coreos/coreos-vagrant.git
```

2.user-data 파일과 config.rb 파일을 만든다.

- user-data: cloud-config 가 구동될때 실행될 서비스를 정의한다.
- config.rb: vagrant 환경을 정의한다.

config.rb

```
$num_instances=1
$update_channel='stable'
$forwarded_ports = { 49153 => 49153 }
$shared_folders = { '/apps' => '/apps' }
$new_discovery_url='https://discovery.etcd.io/new'
```

user-data 1/2

```
#cloud-config

coreos:
  etcd2:
    #generate a new token for each unique cluster from https://discovery.etcd.io/new
    #discovery: https://discovery.etcd.io/<token>
    # multi-region and multi-cloud deployments need to use $public_ipv4
    advertise-client-urls: http://$public_ipv4:2379
    initial-advertise-peer-urls: http://$private_ipv4:2380
    # listen on both the official ports and the legacy ports
    # legacy ports can be omitted if your application doesn't depend on them
    listen-client-urls: http://0.0.0.0:2379,http://0.0.0.0:4001
    listen-peer-urls: http://$private_ipv4:2380,http://$private_ipv4:7001
  fleet:
    public-ip: $public_ipv4
  flannel:
    interface: $public_ipv4
```

user-data 2/2

```
units:
  - name: etcd2.service
    command: start
  - name: fleet.service
    command: start
  - name: flanneld.service
    drop-ins:
    - name: 50-network-config.conf
      content: |
        [Service]
        ExecStartPre=/usr/bin/etcdctl set /coreos.com/network/config '{ "Network": "10.1.0.0/16" }'
    command: start
  - name: docker-tcp.socket
    command: start
    enable: true
    content: |
      [Unit]
      Description=Docker Socket for the API

      [Socket]
      ListenStream=2375
      Service=docker.service
      BindIPv6Only=both

      [Install]
      WantedBy=sockets.target
```

[실습3-25] CoreOS 단일 머신

3. 머신을 실행한다.

```
$ vagrant up
```

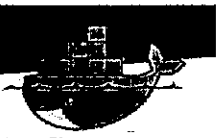
4. 머신 확인

```
$ vagrant status
```

5. 머신 접속

```
$ vagrant ssh core-01
```

[실습3-26] Docker 클라이언트에 CoreOS 설정



Docker client 가 바라보는 Host정보

1. Docker 클라이언트 호스트 설정을 변경한다.

```
$ export DOCKER_HOST='tcp://127.0.0.1:2375'
```

```
$ export DOCKER_TLS_VERIFY=no
```

분산 환경 기반의 key/value 저장소 서버

· 분산 시스템(d)을 위한 /etc 폴더

· http로 접근 가능

· 디렉토리 구조, 파일 시스템

· 변경사항 Watch

분산 환경 기반의 key/value 저장소 서버

· 분산 시스템(d)을 위한 /etc 폴더

· http로 접근 가능

· 디렉토리 구조, 파일 시스템

· 변경사항 Watch

[실습3-26] etcd 실습

1. 값 쓰고 읽기

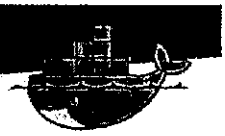
```
$ etcdctl put foo " Hello World! "
```

```
$ etcdctl get foo
$ etcdctl --write-out="json" get foo
```

2. 삭제

```
$ etcdctl del foo
```

[실습3-27] etcd 실습



3. 값 Watch

```
$ etcdctl watch stock1
```

```
$ etcdctl put stock1 1000
```

4. 클러스터 상태

```
$ etcdctl endpoint status
```

https://github.com/coreos/etcd/blob/master/Documentation/demo.md

[실습3-28] Fleet 실습

1. fleetctl 설치

```
$ brew install fleetctl
```

2. fleet 설정

```
export FLEETCTL_DRIVER=API
export FLEETCTL_ENDPOINT=http://localhost:49153
```

3. 머신 목록 확인

```
$ fleetctl list-machines
```

systemd

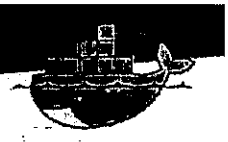
- 유닉스 시스템 관리 데몬, init 을 대체
- 서비스는 unit 파일로 정의
  - service unit, target unit
- 관리도구: systemctl
  - systemctl start | stop | restart <unitname>

unit 파일

· systemd로 실행하는 서비스를 정의

```
[Unit]
Description=OpenSSH server daemon
Documentation=man:sshd(8) man:sshd_config(5)
After=network.target sshd-keygen.service
Wants=sshd-keygen.service

[Service]
EnvironmentFile=/etc/sysconfig/sshd
ExecStart=/usr/sbin/sshd -D $OPTIONS
ExecReload=/bin/kill -HUP $MAINPID
KillMode=process
Restart=on-failure
RestartSec=42s

[Install]
WantedBy=multi-user.target
```


[실습3-29] systemd 실습



1. 서비스 상태 확인

```
$ systemctl status etcd2.service
```

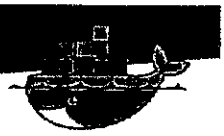
2. 서비스 시작/정지/재시작

```
$ systemctl start etcd2.service
$ systemctl stop etcd2.service
$ systemctl restart etcd2.service
```

3. 부팅시 자동 시작

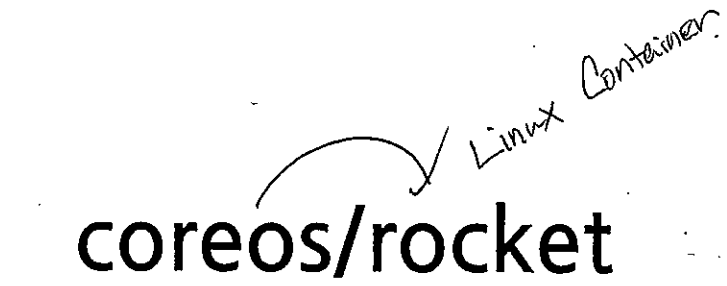
```
$ systemctl enable etcd2.service
```

[실습3-29] systemd 실습



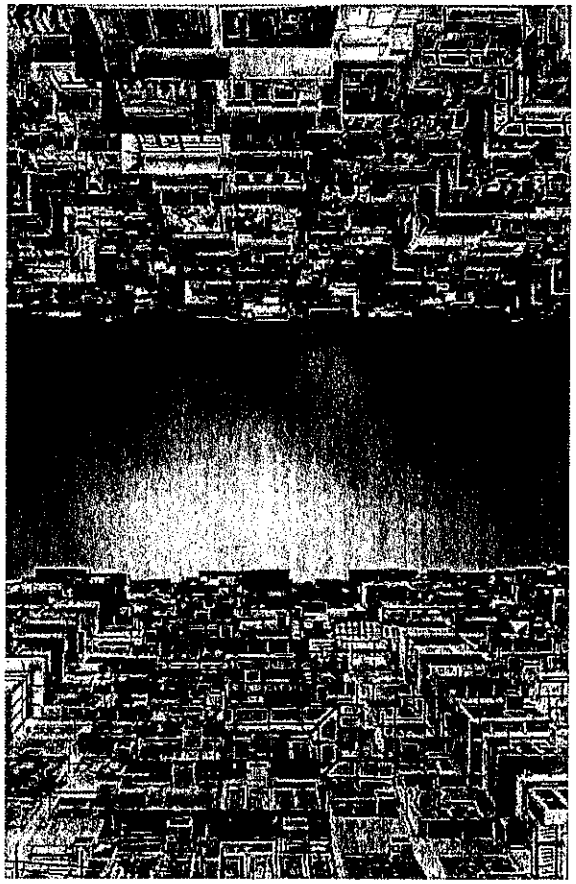
4. 서비스 목록 확인

```
$ systemctl list-unit-files
```



컨테이너 이미지,런타임,디스커버리

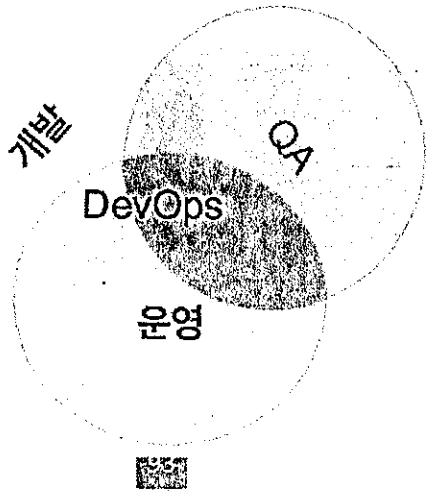
DevOps와 Docker



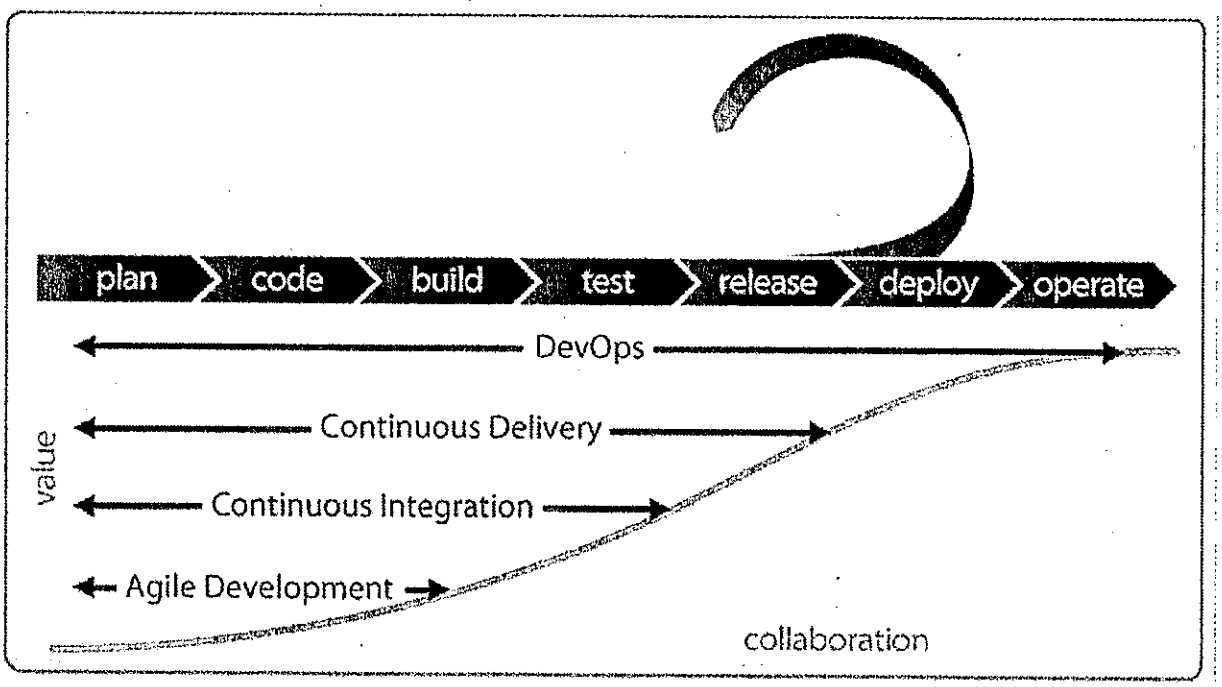
DevOps 환경을 위한 시스템
Docker 오케스트레이션 도구

DevOps 는 무엇인가

서비스나 제품의 릴리즈를 위해서 개발/운영/QA 활동 등이 서로 협업하여 장애없이 서비스나 어플리케이션을 빠르게 릴리즈 하는 방법



Agile, DevOps 뭐가 다른거야



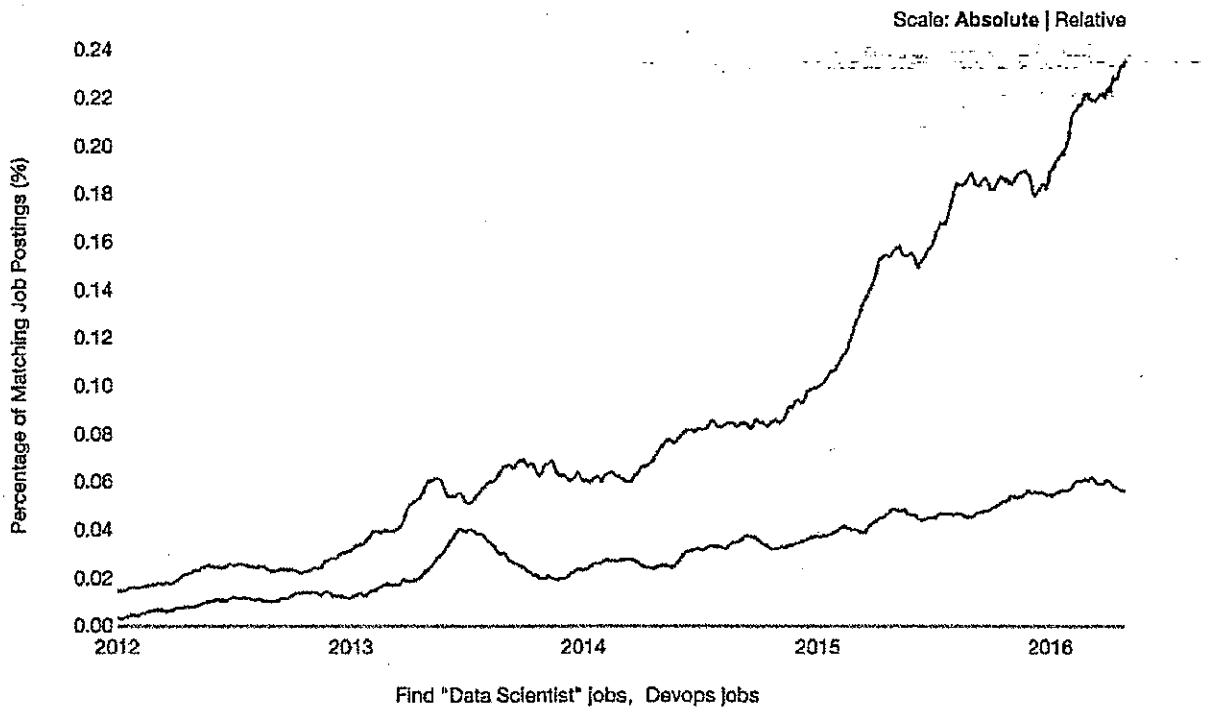
http://www.collab.net/solutions/devops

DevOps도 컨설팅을 위해 만든 용어?

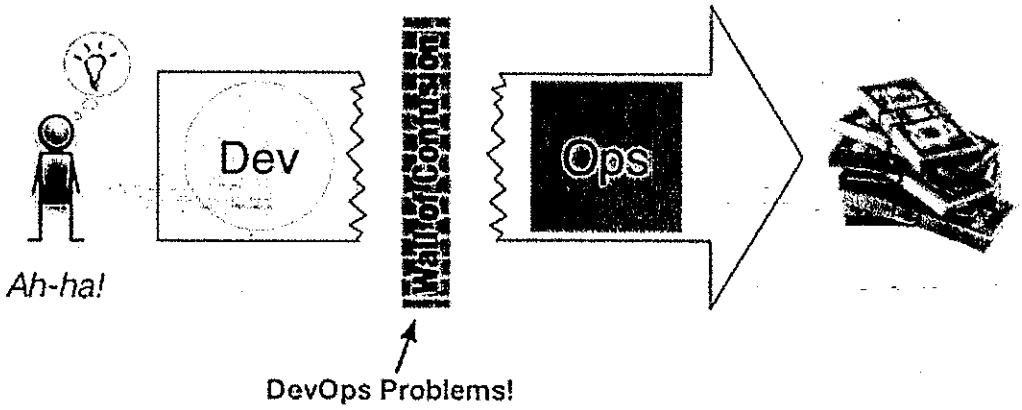
DEVOPSDAYS
DFW·2016

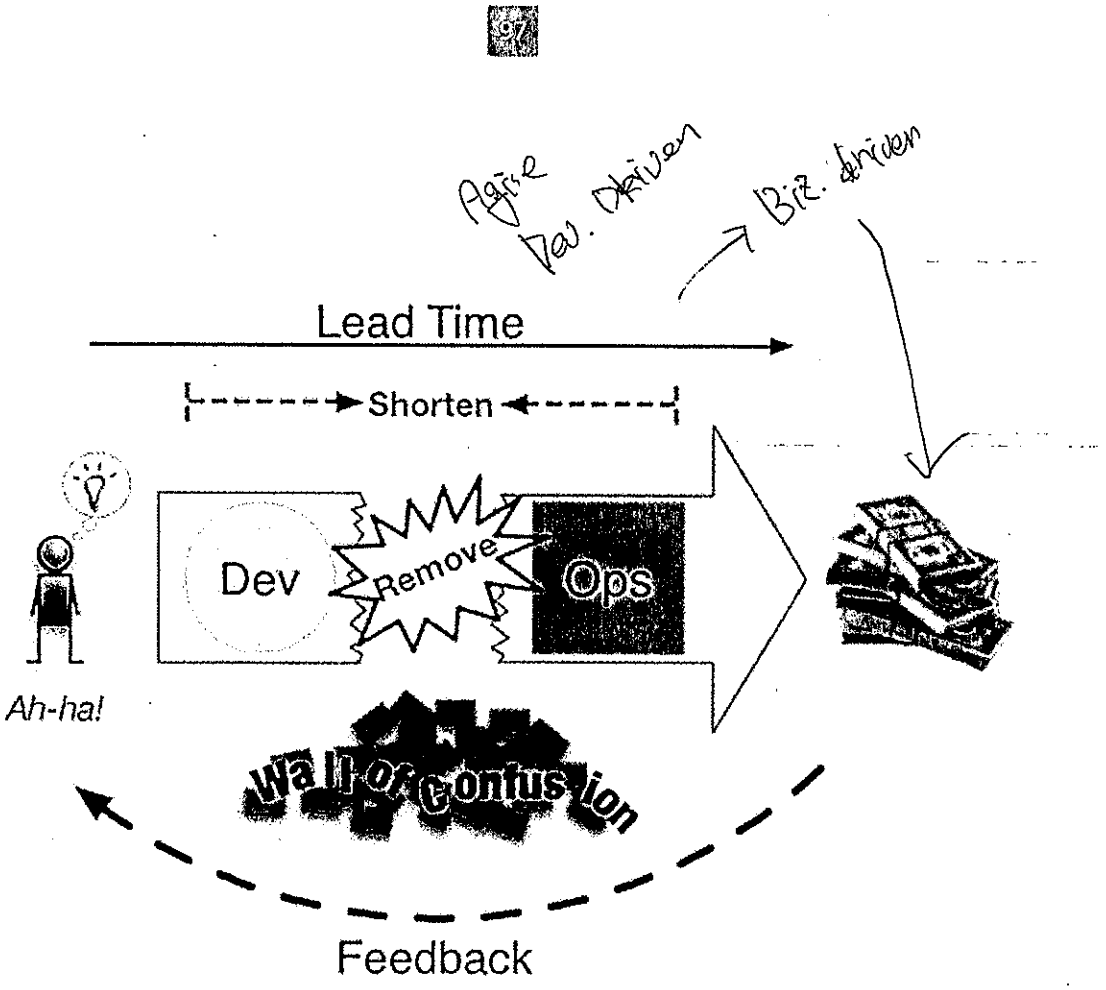
http://www.devopsdays.org/

indeed.com Job Trend



사람들은 왜 DevOps에 열광하나





http://dev2ops.org/2012/09/use-devops-to-turn-it-into-a-strategic-weapon/

누가 DevOps 를 도입했는가

Google facebook.

Etsy NETFLIX

99

Research at Facebook

Home Research Academic Programs **Publications** Blog People

Development and Deployment at Facebook

Dror Feitelson, Eitan Frachtenberg, Kent Beck
IEEE Internet Computing 17(4) · July 1, 2013
Systems Infrastructure, Systems

Abstract

More than one billion users log in to Facebook at least once a month to connect and share content with each other. Among other activities, these users upload over 2.5 billion content items every day.

In this article we describe the development and deployment of the software that supports all this activity, focusing on the site's primary codebase for the Web front-end.

Download Paper

· 엔지니어는 공통 코드베이스에서 일한다.

  · (머지,브랜치 없음)

· 테스트를 전담하는 별도 QA팀은 없다.

· 새로운 코드는 하루에 2번 릴리즈 된다.

Deployment Pipeline

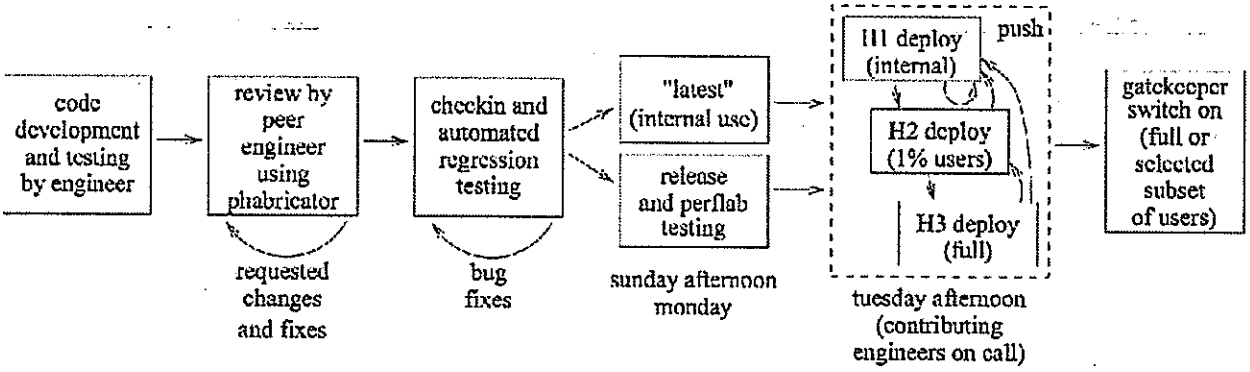
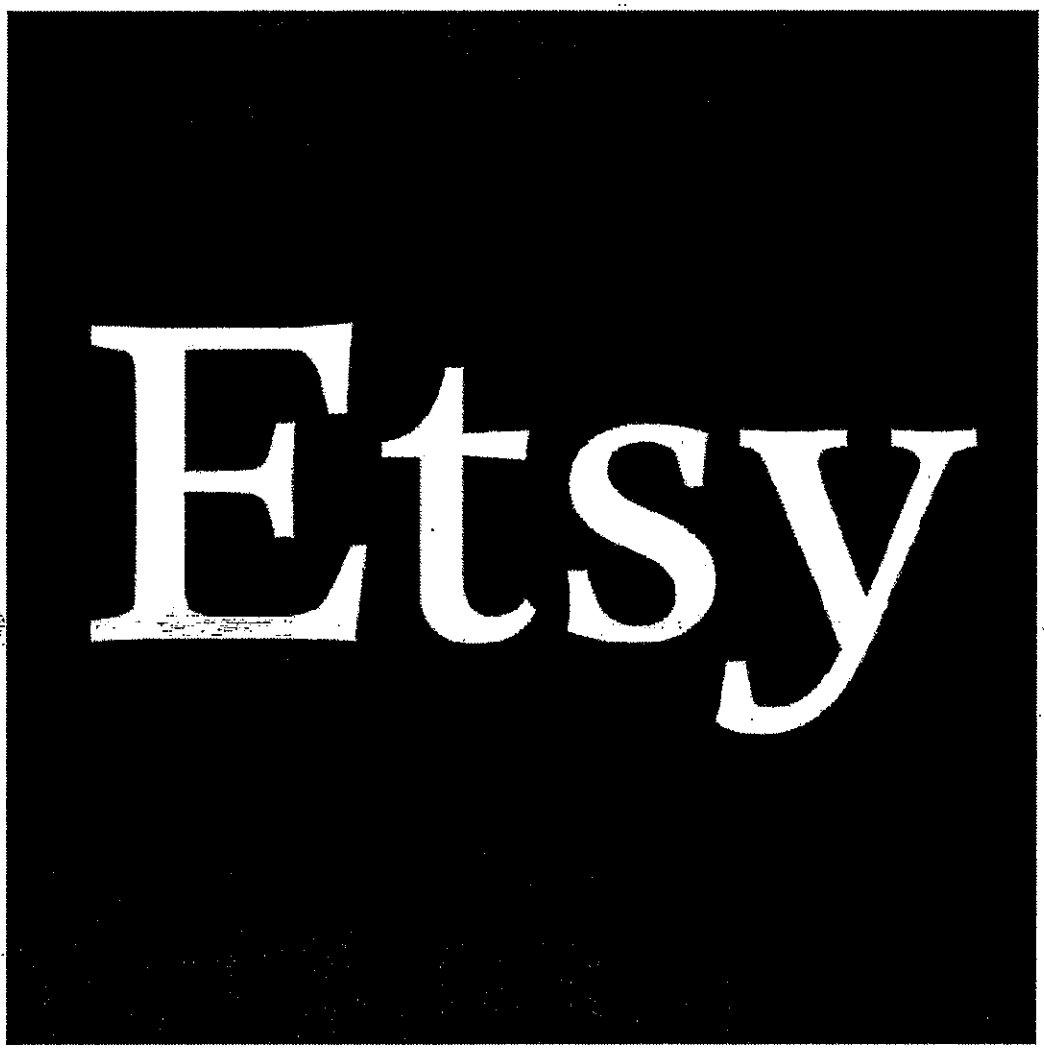


Figure 8: *Facebook's version of the deployment pipeline, showing the multiple controls over new code.*



숫자로 보는 Etsy

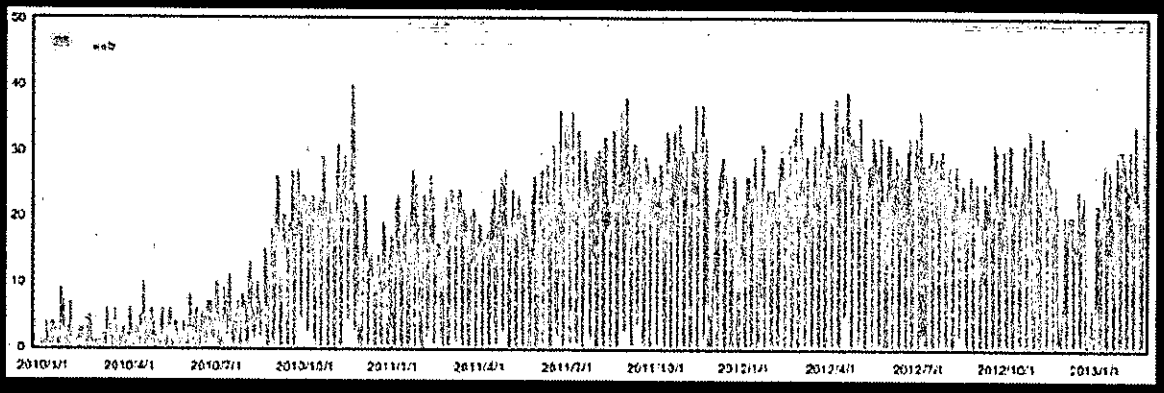
- 사용자수: 5천4백만(2015/3/4)
- 실제 활동하는 구매자 수: 2천5백만(2016/5/3)
- 셀러 수: 1천6백만
- 판매하는 제품수: 3천5백만
- 국가수: 83개국
- 2016년 1분기 매출: $8천1백만

http://expandedramblings.com/index.php/etsy-statistics/

Etsy : Today

1. Small changesets
2. Engineers deploy
3. Deploys are fast
4. Changes behind flags
5. Copious graphs/metrics
6. Fix fast & roll forward

7. Repeat 25+ times per day, every day



Etsy Deploy Stats: 2012

- Deployed to production 6,419 times
- On average, 535/month, 25/day
- Additional 3,851 config-only deploys
- 196 different people deployed to prod
- Nov/Dec 2012: deployed 752 times

누가 DevOps 를 도입했는가

Company	Deploy Frequency	Deploy Lead Time	Reliability	Customer Responsiveness
Amazon	23,000 / day	Minutes	High	High
Google	5,500 / day	Minutes	High	High
Netflix	500 / day	Minutes	High	High
Facebook	1 / day	Hours	High	High
Twitter	3 / week	Hours	High	High
Typical enterprise	Once every 9 months	Months or quarters	Low / Medium	Low / Medium

출처: The Phoenix Project, 2013

카카오톡 사례

- 오픈스택 기반 클라우드 컴퓨팅 인프라의 가상머신(VM)
- 8천개 이상을 단 2명의 전담인력으로 관리하고 있다

http://m.zdnet.co.kr/news_view.asp?article_id=20160627165341

http://superuser.openstack.org/articles/kakaotalk-speaks-volumes-about-the-future-of-cloud-services

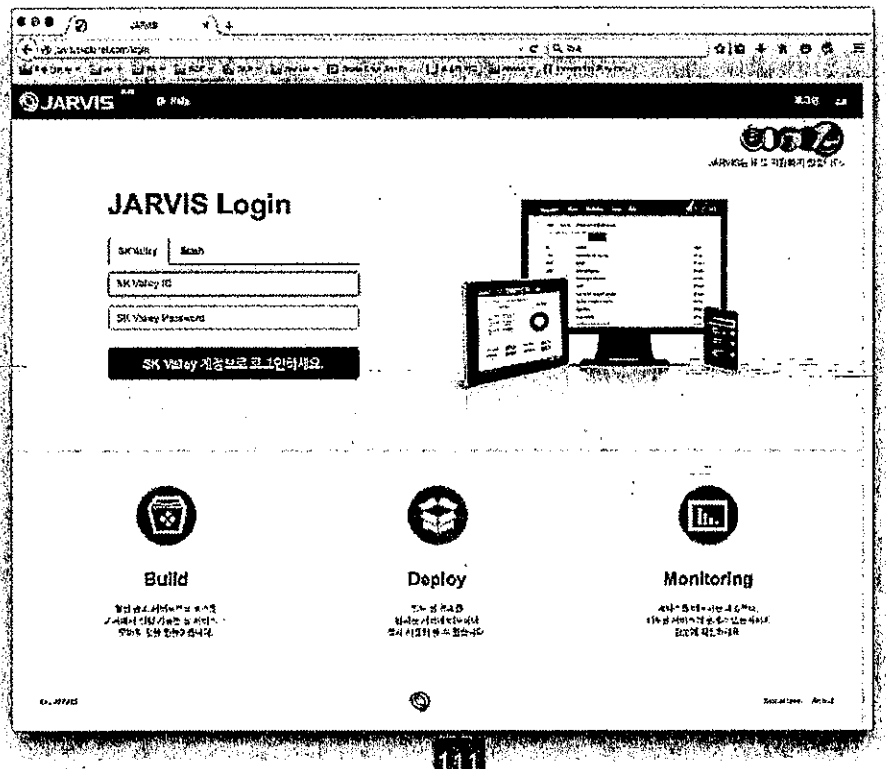
카카오처럼 혼자서 **수천개 VM을 관리할 수 있게 만든 핵심 요소**는 뭐였을까. 공 수석에 따르면 그건 기술이 아니라 '문화'에 달렸다. 공 수석은 자신들의 조직문화가 '**자동화(Automation)**'고, 기술은 이 문화를 지원할 수 있는 수단이라고 설명했다.

공 수석은 같은 맥락에서 클라우드를 기술이 아니라 "프로그래밍할 수 있는 자원관리(Programmable Resource Management)"라는 방법론적 개념으로 정의하고 이를 실현하기 위해 클라우드 데이터센터가 갖춰야 하는 기능 3가지를 제시했다. 중앙집중형 구성관리데이터베이스(Centralized CMDB), 중앙집중형 측정 시스템(Centralized Measuring system), **중앙 배포 시스템(Central Deploying system)**이다.

홈쇼핑처럼 사례

http://subicura.com/2016/06/07/zero-downtime-docker-deployment.html

JARVIS(빌드/배포 시스템)



JARVIS(빌드/배포 시스템)

- 프로젝트 기간: 2013/4~12
- SK Planet 표준 배포시스템
- 배포하는 서버 댓수: 1400
- 빌드 누적횟수: 48,000(230건/일)

http://readme.skplanet.com/?p=7148

JARVIS 워크플로

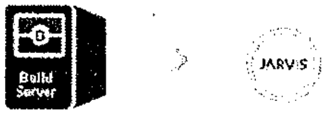
Step1

소스코드 레파지토리를 임포트 한다.
- 관리 프로젝트를 형상관리 서버로부터 입력 받아 빌드/배포 프로젝트를 만든다.
- SK Valley에 등록된 프로젝트는 별다른 작업없이 'import'버튼만 클릭하면 레파지토리를 가져온다.

JARVIS Project 025	OFF	OFF
JARVIS Project 028	OFF	OFF
JARVIS Project 032	OFF	OFF

Step2

빌드 환경을 설정한다.
- 사용하고 있는 빌드서버를 등록하면 등록된 빌드 프로젝트로 임포트 한다.
- SK Valley 레파지토리에 대한 빌드 프로젝트가 없다면 자동으로 빌드환경을 설정해 준다.

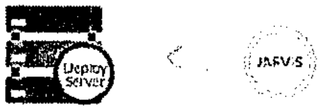


JARVIS Project 025	ON	OFF
JARVIS Project 028	ON	OFF
JARVIS Project 032	OFF	OFF

118

Step3

배포 환경을 설정한다.
- 배포하고자 하는 타겟서버를 등록하면 타겟서버에 에이전트를 설치 해준다.
- 이제 배포할 경로를 결정하고 승인여부를 확인하여 배포 환경을 설정해 준다.



JARVIS Project 025	ON	ON
JARVIS Project 028	ON	OFF
JARVIS Project 032	OFF	OFF

Step4

[illegible] 빌드하고 배포한다.
- 빌드 완료 성공이나 배포 완료 성공이 설정된 프로젝트는 JARVIS를 이용해 즉시 배포할 수 있다.

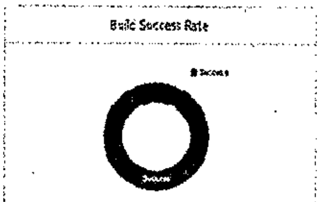
JARVIS Project 025	ON	ON
JARVIS Project 032	ON	OFF

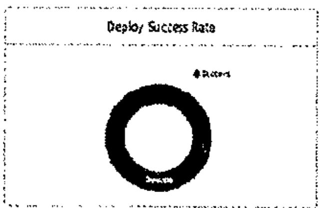
Step5

진행 결과를 모니터링 한다.
- 빌드/배포가 정상적으로 진행되었는지 확인하고, 문제가 발생하면 즉시 담당자에게 알려준다.

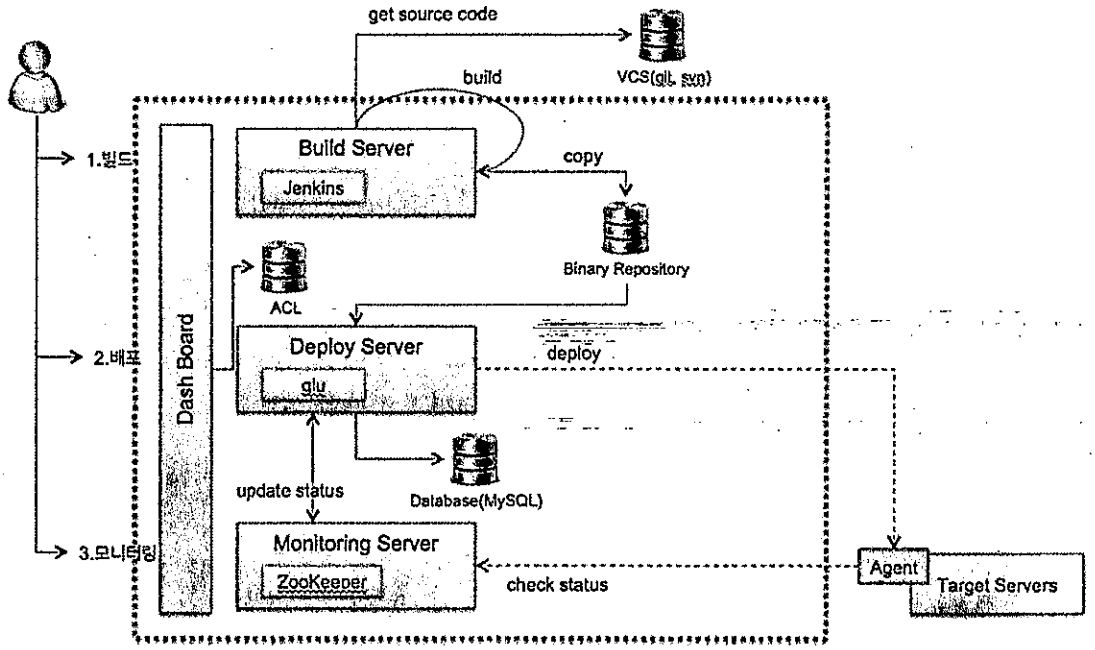
Information

Type	Status	Date
Build	Success	2013-11-08 08:06:30
Deploy	No. Data	





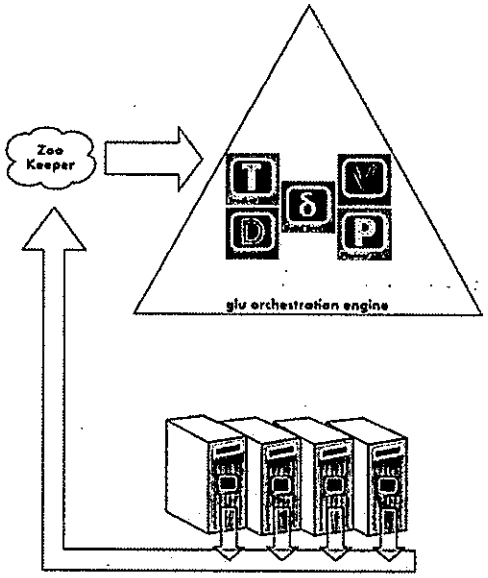
JARVIS 모듈 아키텍처



배포 솔루션

이름	라이센스	특징
Control Tier	오픈소스	구버전, 단종
Glu	오픈소스	Linkedin 개발
System Center	상용	마이크로소프트 솔루션
uDeploy	상용	기능은 다양
Bamboo	상용	Atlassian 솔루션

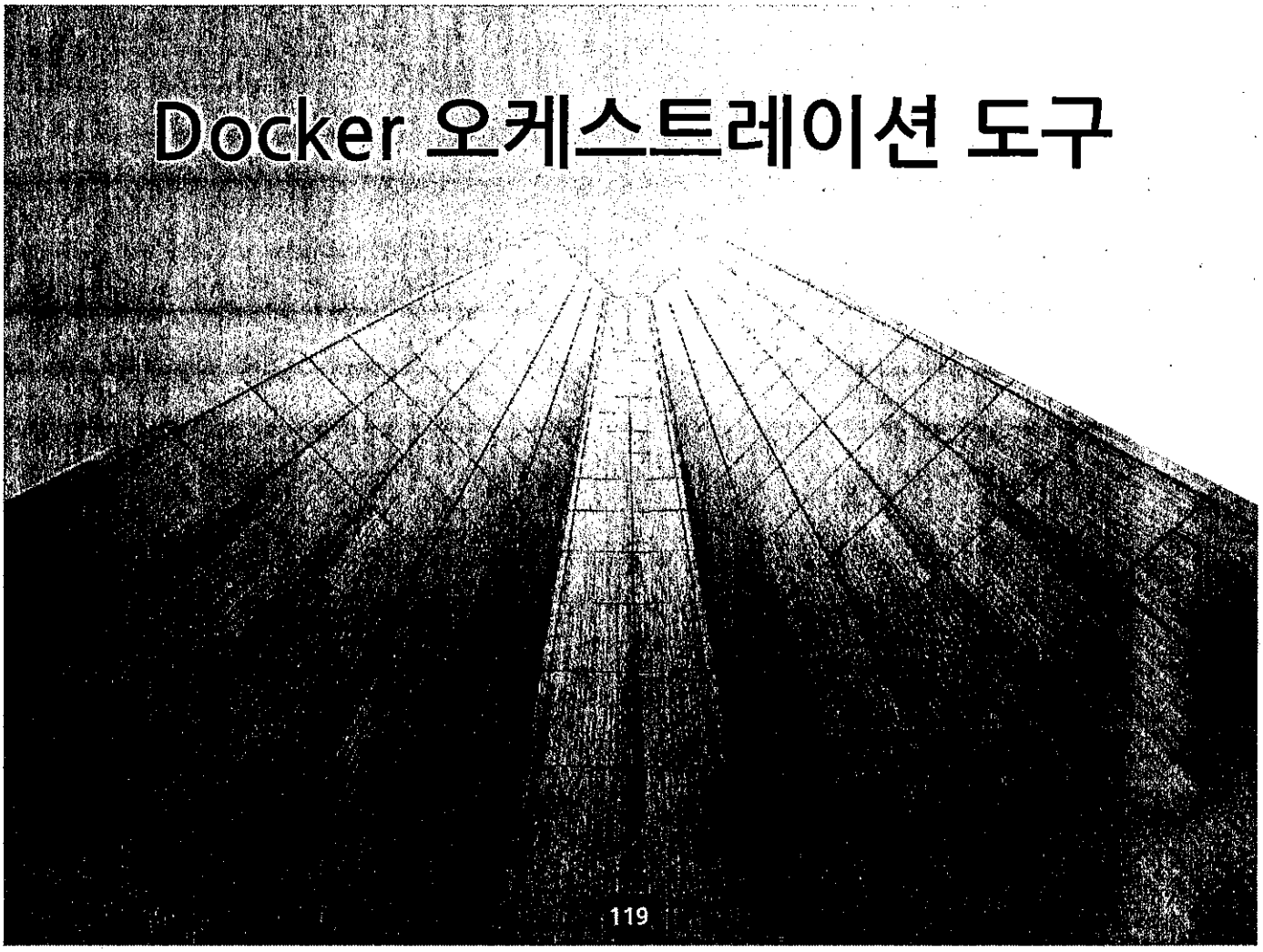
glu 시스템 구성



DevOps를 위해 필요한것들

- 자동화(Automaiton)
  - Provisioning, Deployment, Test Automation
- 문화(Culture)
  - Cross functional team
- 개인
  - 프로그래밍 능력

Docker 오케스트레이션 도구



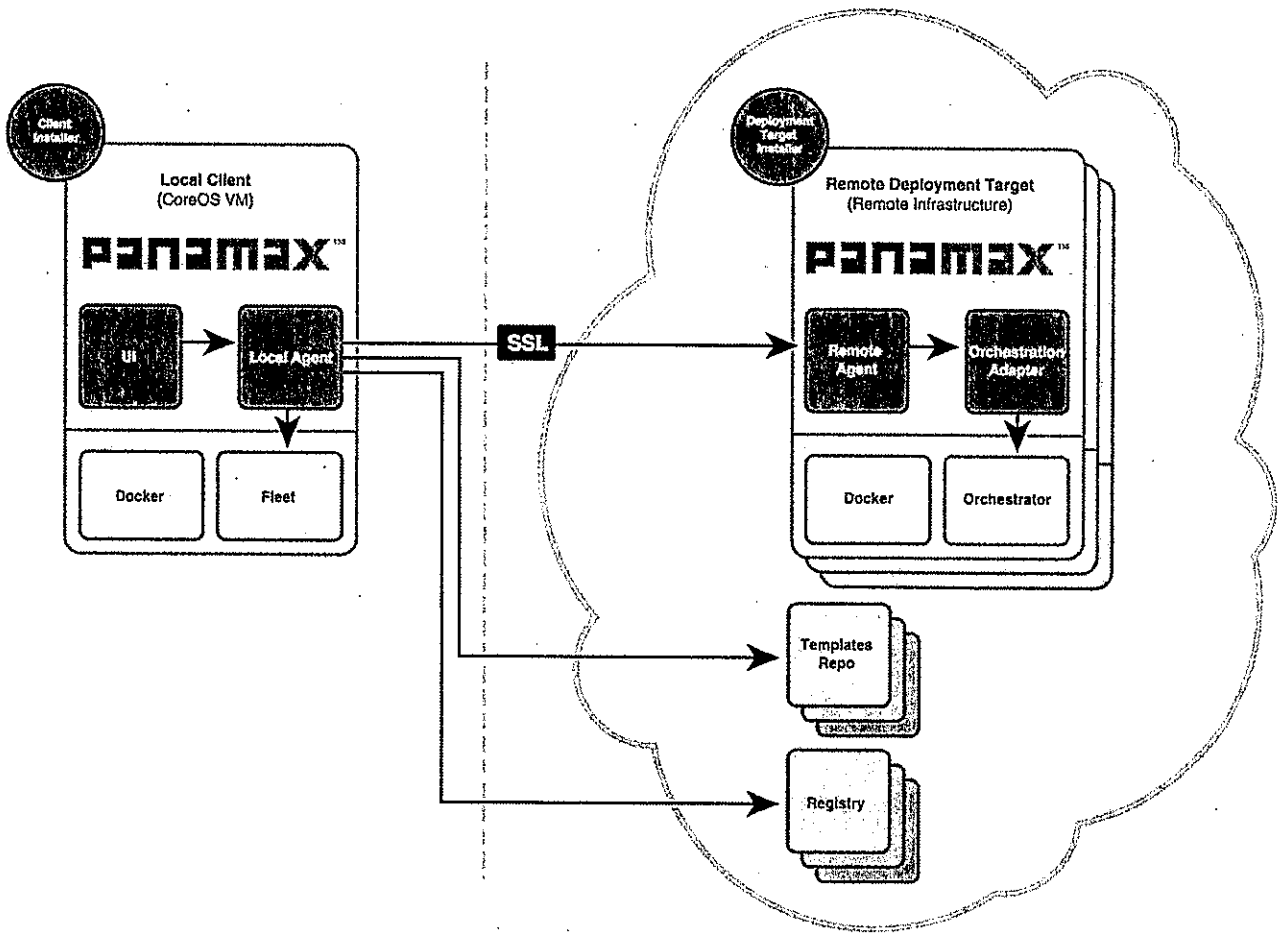
Docker 오케스트레이션

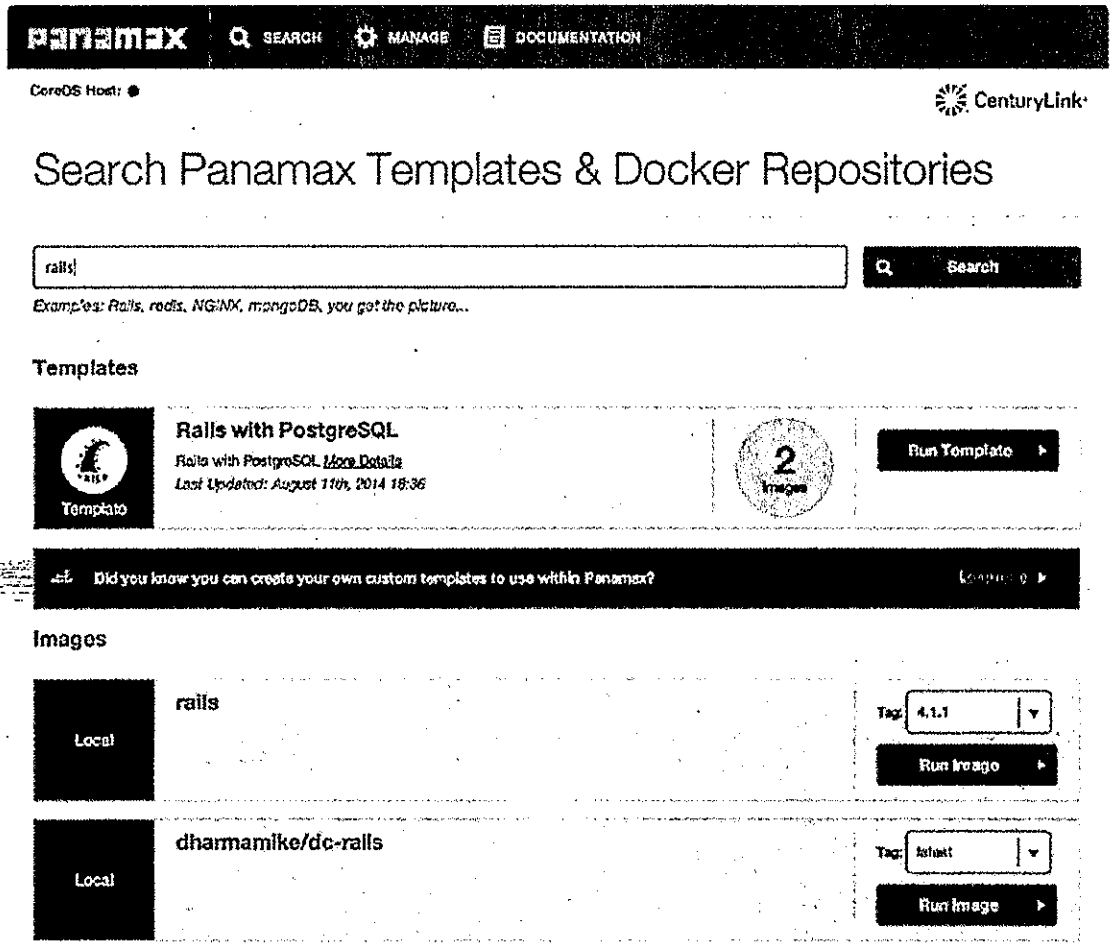
- Panamax
- spotify/helios
- kubernetes
- Coreos/fleet
- Rancher

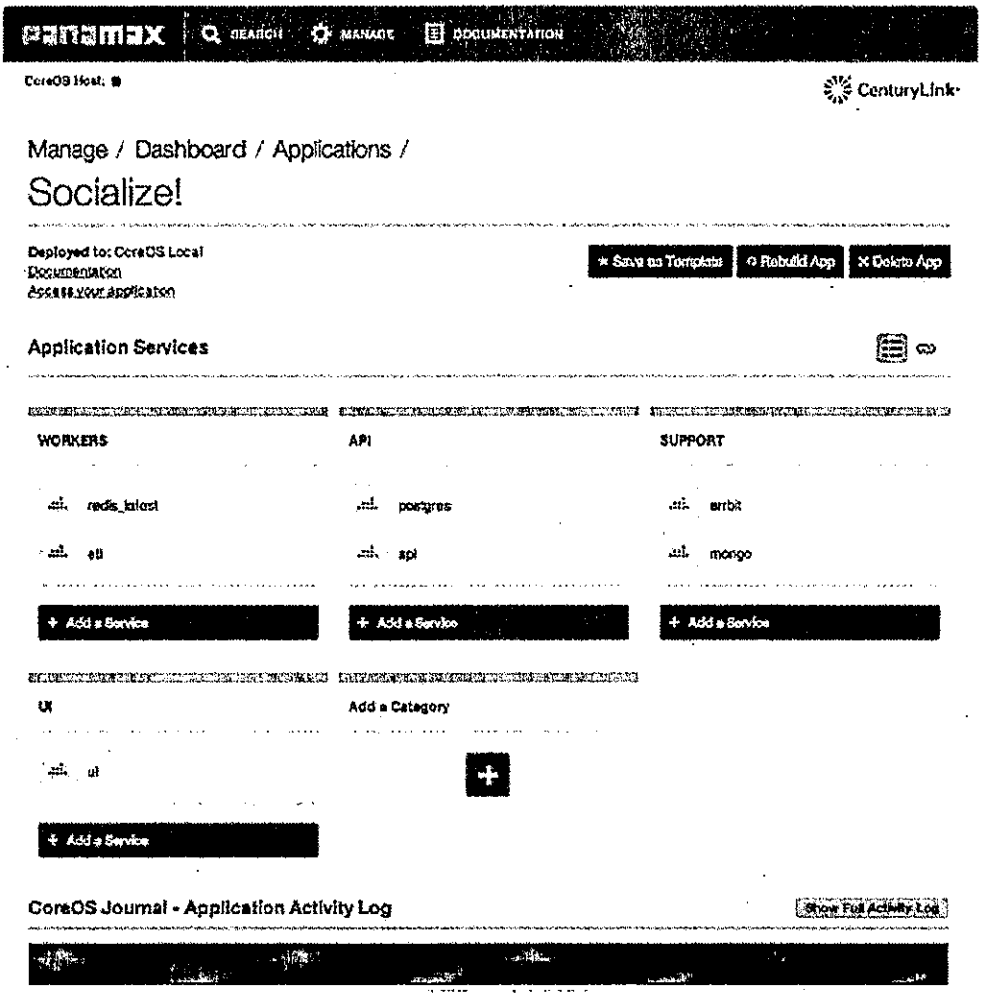
Panamax

도커 컨테이너 관리도구

- RoR, CoreOS, github 기반
- Template 개념, 편리한 UI
- http://panamax.io




panamax
SEARCH
MANAGE
DOCUMENTATION
CoreOS Host:
CenturyLink
Search Panamax Templates & Docker Repositories
rails
Search
Examples: Rails, redis, NGINX, mongoDB, you get the picture...
Templates
Rails with PostgreSQL
Rails with PostgreSQL More Details
Last Updated: August 11th, 2014 18:36
Template
2
Images
Run Template
Did you know you can create your own custom templates to use within Panamax?
Images
Local
rails
Tag: 4.1.1
Run Image
Local
dharmamike/dc-rails
Tag: latest
Run Image

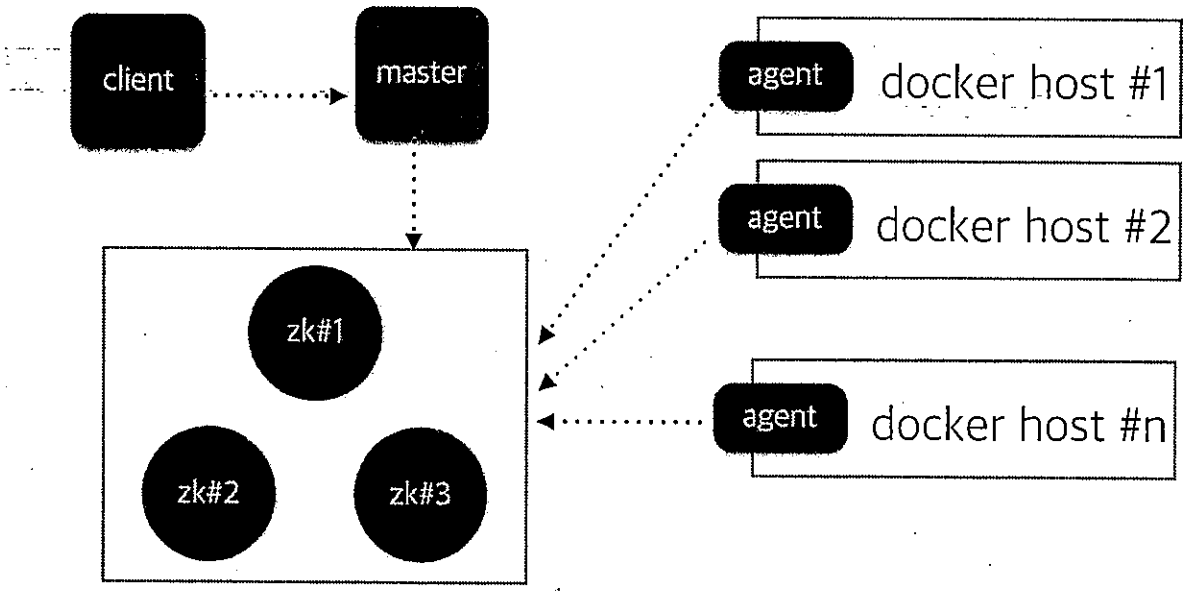

panamax
SEARCH
MANAGE
DOCUMENTATION
CoreOS Host:
CenturyLink
Manage / Dashboard / Applications /
Socialize!
Deployed to: CoreOS Local
Documentation
Access your application
Save as Template
Rebuild App
Delete App
Application Services
WORKERS
API
SUPPORT
redis_latest
postgres
errbit
elk
api
mongo
Add a Service
Add a Service
Add a Service
UI
Add a Category
ui
Add a Service
CoreOS Journal - Application Activity Log
Show Full Activity Log

spotify/helios

도커 이미지,컨테이너 라이프사이클 관리

- Java, CLI
- Job 단위로 처리, 스케쥴링 X
- https://github.com/spotify/helios

spotify/helios

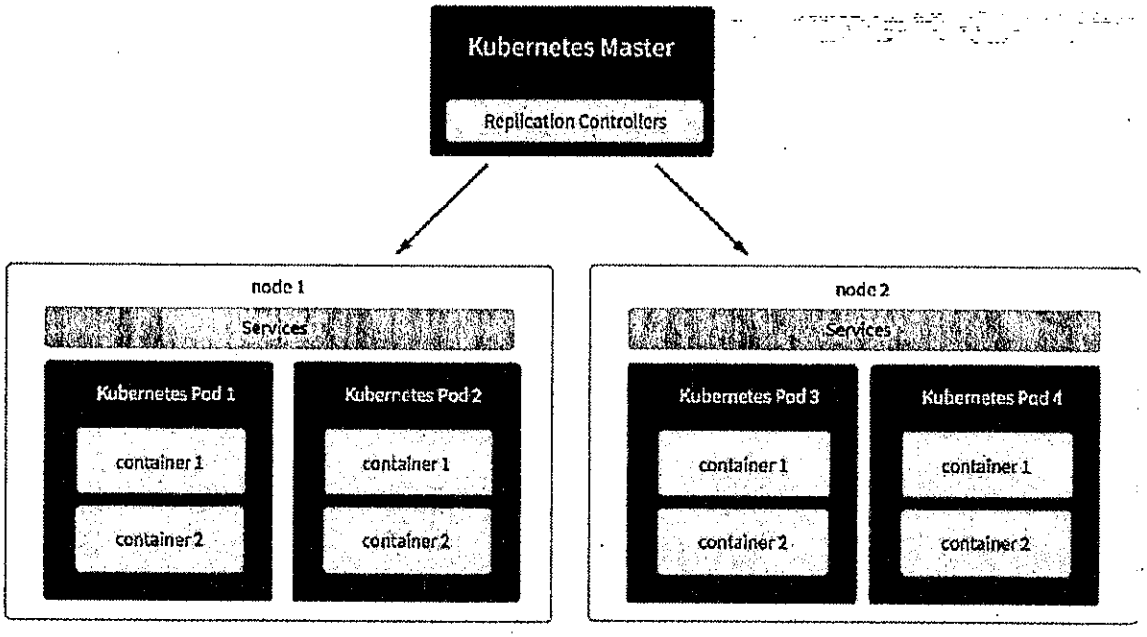


kuberntes(k8s)

- 구글에서 만든 리눅스 컨테이너 관리 시스템
- 클라이어트: kubectl, dsahboard
- 공식 사이트: http://kubernetes.io/

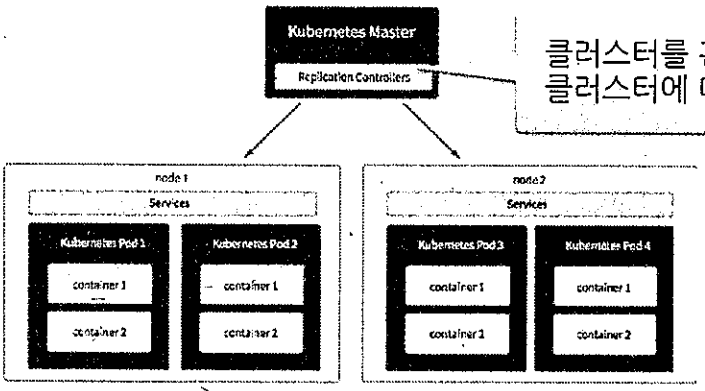
kubernetes
by Google

k8s 구성요소



k8s 구성요소

k8s Master



클러스터를 관리하고, 클러스터에 대한 API를 호출한다.
클러스터에 대한 모든 것을 관리한다.

node

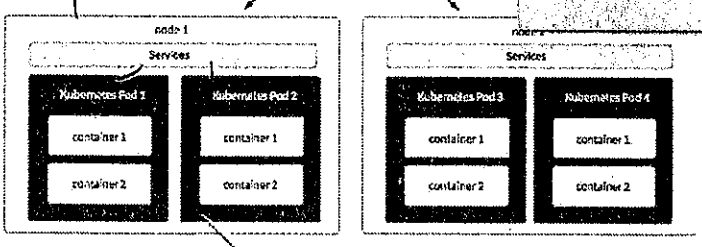
클러스터를 구성하는 하나의 물리 서버(또는 가상 머신), aka Minions
Master의 명령에 따라 pods을 생성하고 관리(실행/중지/삭제 등)한다.

k8s 구성요소

rc

클러스터 내에 지정된 수 만큼의 pod이 실행되는 것이 보장될 수 있도록 관리

생성된 pods에 문제가 발생하면 새로운 pods을 필요한 수 만큼 만들고, 문제가 해결되면 넘치는 pods을 없앰.

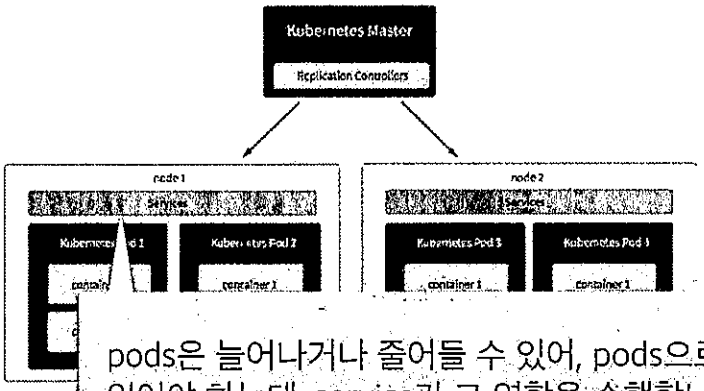


POD

k8s에서 기본 생성 단위로, 컨테이너의 집합

CPU, memory, volumes, IP address 및 기타 자원을 할당. (pod 내의 모든 컨테이너가 자원을 공유)

k8s 구성요소



Service

pods은 늘어나거나 줄어들 수 있어, pods으로 가는 traffic을 load balancing 해 줄 수 있어야 하는데, service가 그 역할을 수행합니다.

Private IP (또는 Public IP)가 service에 할당되고

Service에 할당된 "IP + port"를 통해 접근하면, 뒤에 있는 pods에 round-robin 등의 방식으로 접근

sticky session 지원

[실습29] Minikube 설치

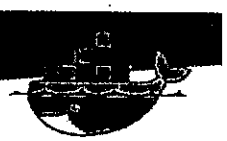
1. minikube 설치

```
$ curl -Lo minikube https://
storage.googleapis.com/minikube/releases/latest/
minikube-darwin-amd64 && chmod +x minikube &&
sudo mv minikube /usr/local/bin/
```

https://kubernetes.io/docs/tutorials/stateless-application/hello-minikube/

https://kubernetes.io/docs/getting-started-guides/minikube/

[실습29] Minikube 설치



2. kubectl 설치

```
$ curl -LO https://storage.googleapis.com/
kubernetes-release/release/$(curl -s https://
storage.googleapis.com/kubernetes-release/
release/stable.txt)/bin/darwin/amd64/kubectl
$ chmod +x ./kubectl
$ sudo mv ./kubectl /usr/local/bin/kubectl
```

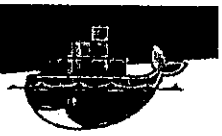
3. minikube 시작

```
$ minikube start --vm-driver=virtualbox
```

4. 클러스터 확인

```
$ kubectl cluster-info
```

[실습30] hello-node 생성



```
$ eval $(minikube docker-env)

$ docker build -t hello-node:v1 .
```

[실습31] hello-node 배포

1. hello-node 배포

```
$ kubectl run hello-node --image=hello-node:v1 --
port=8080
```

2. 배포(deployments) 확인

```
$ kubectl get deployments
```

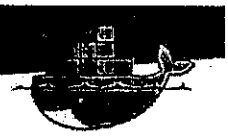
3. pods , events 확인

```
$ kubectl get pods
```

4. 설정 확인

```
$ kubectl config view
```

[실습32] 서비스 생성



1. hello-node 서비스 생성

```
$ kubectl expose deployment hello-node --
type=LoadBalancer
```

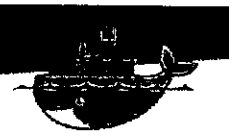
2. 서비스 확인

```
$ kubectl get services
```

3. hello-node 테스트

```
$ minikube service hello-node
```

[실습33] 롤링 업데이트



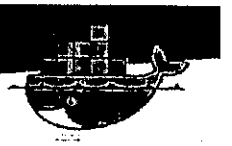
1. hello-node:v2 이미지 생성

```
$ docker build -t hello-node:v2 .
```

2. 업데이트

```
$ kubectl set image deployment/hello-node hello-node=hello-node:v2
```

[실습34] kubenets dashboard

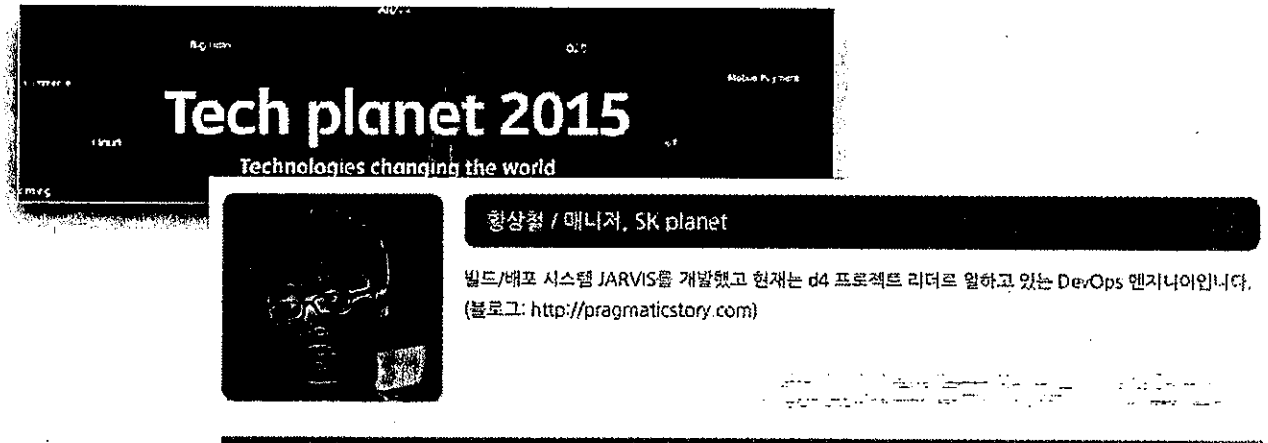


1. 브라우저로 dashboard에 접속한다.

http://<<kubernets-master>>/ui

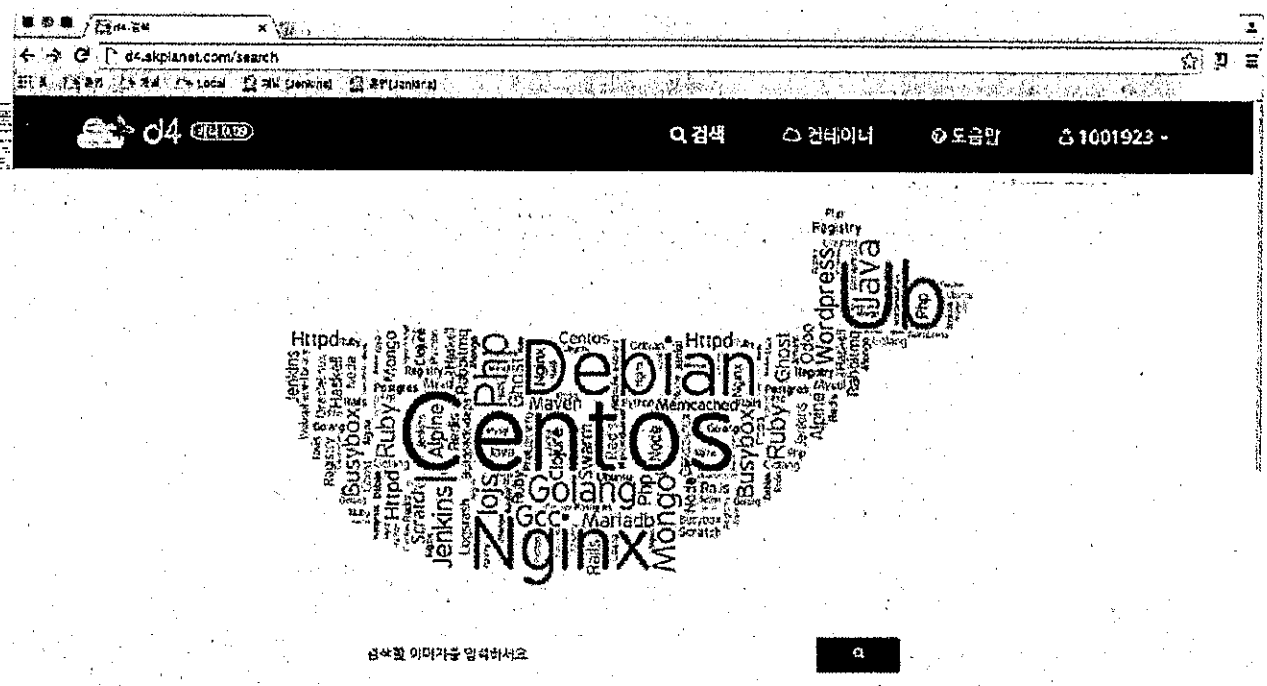
https://github.com/kubernetes/dashboard

Docker 기반 클라우드 환경 구축 사례

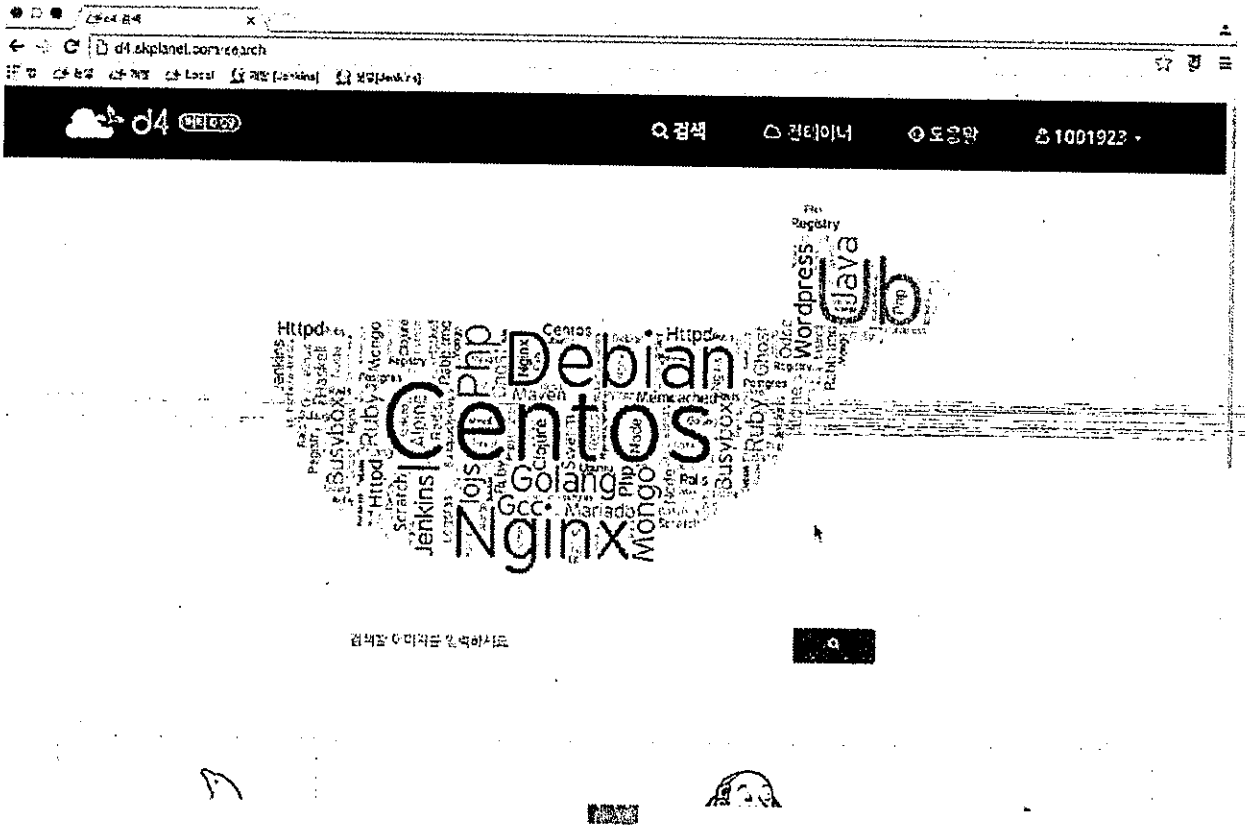


http://techplanet.skplanet.com/speaker_track1.html#track1_7

d4 - jenkins & nginx

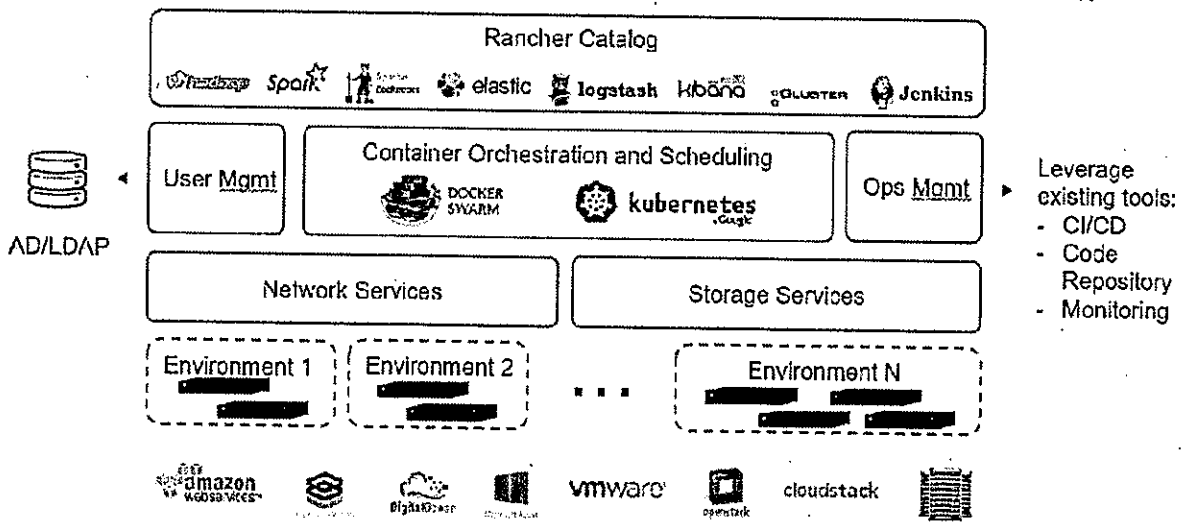


d4 - springboot



Rancher

컨테이너 인프라스트럭처 관리도구, 컨테이너 OS

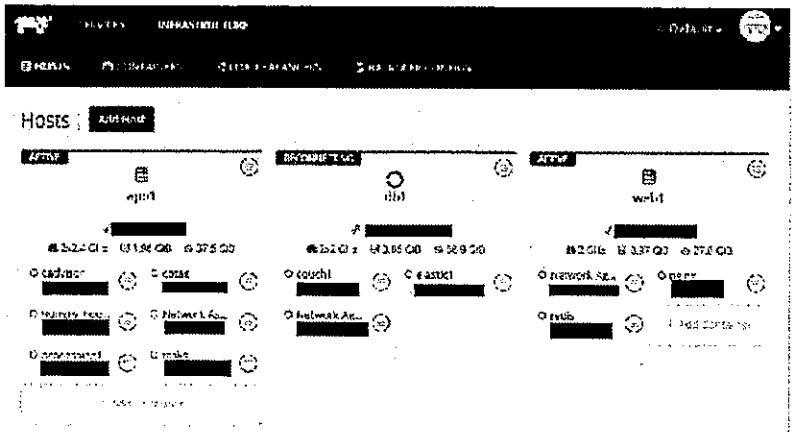


Rancher Key Features

- 멀티 호스트 네트워킹: Private + Public Provider
- 컨테이너 로드밸런싱: HA Proxy 기반
- 스토리지 서비스: Docker1.9 Volume 호환
- 서비스 디스커버리: 분산 DNS 기반, 헬스체크(Agent+Haproxy)
- 서비스 업그레이드: 롤링 업데이트
- 리소스 관리: 리소스 모니터링 및 스케쥴링
- 사용자 관리: LDAP 연동
- 멀티 오케스트레이션 엔진: Cattle, k8s, Swarm

Rancher Interface

- Docker CLI
- rancher-compose: rancher-compose.yml
- Rancher UI



주요 컨셉

- Service: 어플리케이션
- Stacks
  - docker-compose 프로젝트와 유사
  - 서비스 그룹
- Catalog: 어플리케이션 템플릿
  - Rancher certified, community-catalog, Private catalog

추천 자료

- Rancher Monthly Training - June 2016
  - https://www.youtube.com/watch?v=JqcpbI090Is

[실습35] Rancher 설치

1. rancher-host 머신 생성

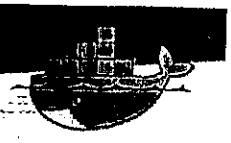
```
$ docker-machine create -d virtualbox rancher-
host
```

2. rancher 서버 실행

```
$ docker run -d --restart=always -p 8080:8080
rancher/server:stable
```

https://hub.docker.com/r/rancher/server/

[실습36] Rancher - Custom 호스트 추가

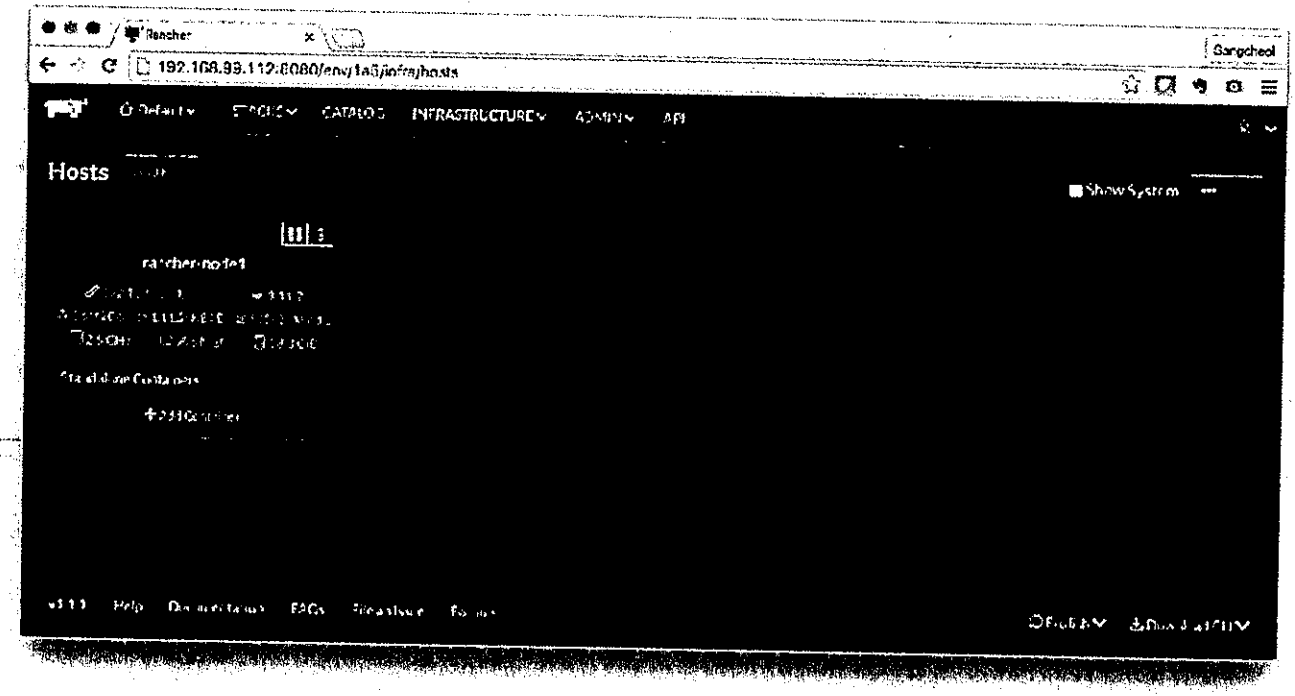


1. rancher-node1 머신 생성

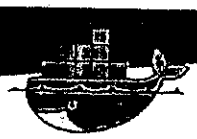
```
$ docker-machine create -d virtualbox rancher-
node1
```

2. rancher agent 설치

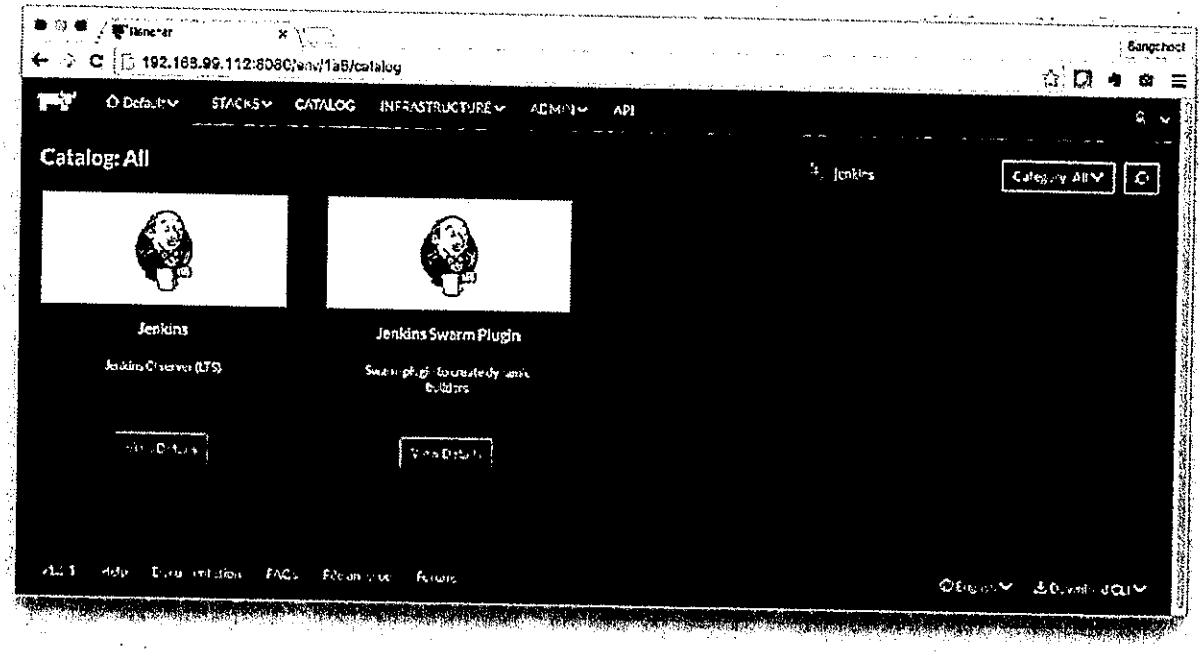
```
$ docker run -d --privileged -v /var/run/
docker.sock:/var/run/docker.sock -v /var/lib/
rancher:/var/lib/rancher rancher/agent:v1.0.2
http://192.168.99.112:8080/v1/scripts/
3CD461AEE5333B39D48E:
1469196000000:YzvVGzFWJWGsitMGS67Dlslww
```

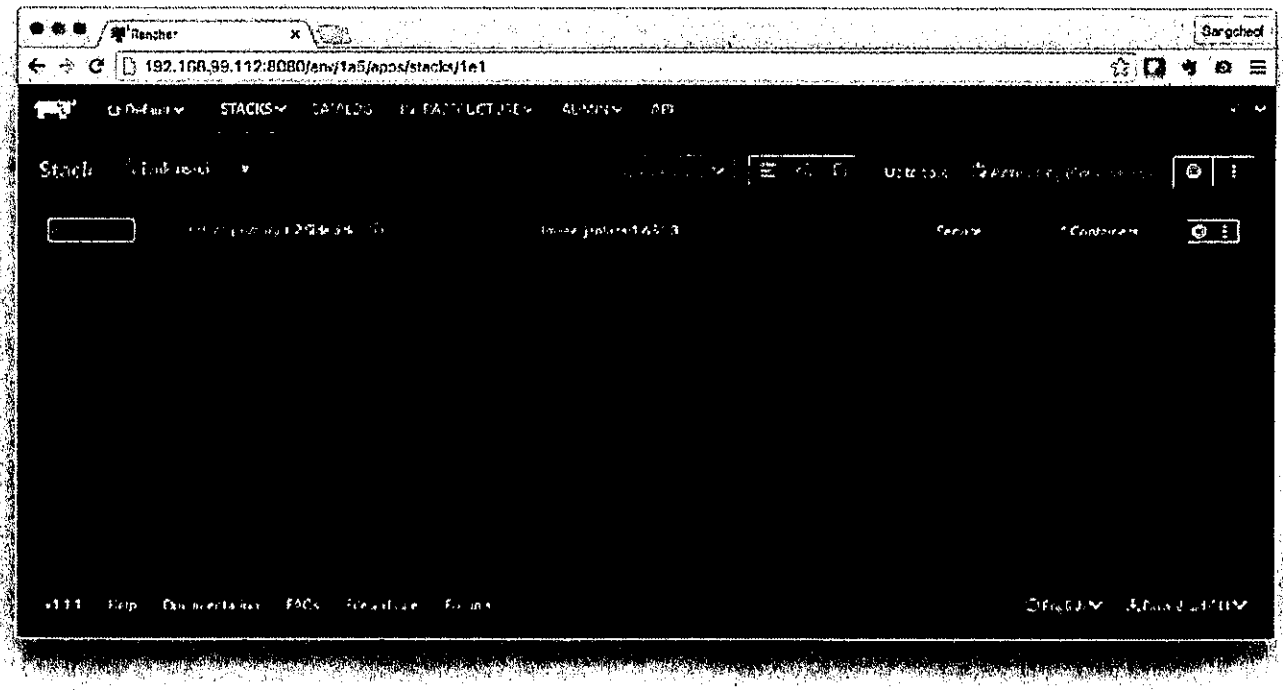


[실습37] Rancher - Catalog 추가



1. Jenkins 추가





Q+A

espressobook