

유엔진 클라우드 운영 가이드

2017 년 1월 18일 유엔진클라우드 박승필 작성 v1.0

- 설치
 - [서버 준비 \(pre-server.md\)](#)
 - [도메인 & 네트워크 준비 \(pre-domain.md\)](#)
 - [네트워크 고정 아이피\(옵셔널\) \(pre-static-ips.md\)](#)
 - [디스크 마운트\(옵셔널\) \(pre-disk.md\)](#)
 - [유틸리티 설치 \(install-util.md\)](#)
 - [클러스터 설치 \(install-cluster.md\)](#)
 - [클러스터 계정 및 CLI \(install-cluster-user.md\)](#)
 - [깃랩 & 도커 레지스트리 설치 \(install-gitlab.md\)](#)
 - [CI 서버 설치 \(install-ci.md\)](#)
 - [클라우드 패키지 빌드 \(install-package.md\)](#)
 - [클라우드 앱 실행 \(install-package-run.md\)](#)
 - [테스트 \(install-test.md\)](#)
 - [cadvisor proxy \(cadvisor.md\)](#)
- 운영
 - [노드 추가 \(op-add-node.md\)](#)
 - [노드 삭제 \(op-remove-node.md\)](#)
 - [설치 완전 삭제 \(op-uninstall.md\)](#)
 - [깃랩 백업 \(op-gitlab-backup.md\)](#)
- [트러블 슈팅 \(trouble.md\)](#)
 - [컴포넌트 헬스 체크 \(trouble-component.md\)](#)
 - [네트워크 \(trouble-network.md\)](#)

서버 준비

사전 준비 사항

— 모든 노드는 CentOS/Rhel 7.2 ~ 7.3 OS 에서 구동됩니다.

- 내부망 UDP 뚫려있을것: UDP 가 뚫려있지 않으면 53 포트 domain resolve 가 작동하지 않는다. (운영시)
- 내부망 TCP 포트는 모두 뚫려있을것 (운영시)
- 외부망 인바운드 : 퍼블릭 에이전트 머신의 80 포트와 443 포트 (운영시)
- 외부망 아웃바운드 : 모두 뚫려있을것 (설치시)

- 네임서버 : *. 형식으로, A MASK 가 * 로 설정된 도메인을 보유하고있을것. (운영시)
- 모든 서버는 동일한 pem 파일로 ssh 가 가능할 것. (운영시)
- 모든 서버의 /etc/ssh/ssh_config 는 PermitRootLogin (설치시) 을 허용할 것.

서버 준비

- 다음의 서버들을 준비해야 합니다. (사양은 추천사항입니다.)
- 마스터 노드는 홀수개의 서버로 준비하도록 합니다. (1,3,5)
- 프라이빗 에이전트 노드는 최소 한개를 구성해야 합니다.
- 퍼블릭 에이전트 노드는 최소 한개를 구성해야 합니다.
- 퍼블릭 에이전트 노드의 80 포트와 443 포트, 그리고 9000 - 60000 (허용 최대치) 포트는 외부에서 접속이 가능해야 합니다.
- 마스터 노드중 한개의 80 포트는 외부에서 접속이 가능해야 합니다.
- 깃랩 서버의 80 포트와 5000 포트는 외부에서 접속이 가능해야 합니다.
- 깃랩 서버, 퍼블릭 에이전트 노드, 그리고 한개의 마스터 노드는 인터넷 환경에서 접속 가능한 퍼블릭 아이피를 가지고 있어야 합니다.

서버 예시

역할 / 호스트네임	사양	IP 주소	퍼블릭 IP 주소	외부 포트바인딩
bootstrap	2 CPU / 2 GB / 100 GB Disk	172.31.8.143		
master1	2 CPU / 4 GB / 100 GB Disk	172.31.12.143	52.79.125.242	80
master2	2 CPU / 4 GB / 100 GB Disk	172.31.4.125		
master3	2 CPU / 4 GB / 100 GB Disk	172.31.1.198		
public-agent	4 CPU / 8 GB / 100 GB Disk	172.31.5.136	52.79.51.79	80,443,9000 - 60000
agent1	4 CPU / 8 GB / 100 GB Disk	172.31.6.35		
agent2	4 CPU / 8 GB / 100 GB Disk	172.31.1.235		
agent3	4 CPU / 8 GB / 100 GB Disk	172.31.5.245		
agent4	4 CPU / 8 GB / 100 GB Disk	172.31.14.247		
agent5	4 CPU / 8 GB / 100 GB Disk	172.31.7.160		
agent6	4 CPU / 8 GB / 100 GB Disk	172.31.11.70		
agent7	4 CPU / 8 GB / 100 GB Disk	172.31.0.164		
gitlab	4 CPU / 32 GB / 300 GB Disk	172.31.15.249	52.78.60.43	80,5000
ci	1 CPU / 1 GB / 100 GB Disk	172.31.3.61		

도메인 && 네트워크 준비

도메인 준비

다음의 도메인들을 준비하도록 합니다. 다음의 보기에서는 pas-mini.io 도메인을 소유했다고 가정하고, 다음의 A_Mask 들을 준비하도록 합니다.

- gitlab, dcos 을 제외한 모든 도메인은 public-agent 서버로 연결토록 합니다.
- gitlab 도메인은 gitlab 서버로, dcos 도메인은 마스터 노드 중 하나로 연결토록 합니다.
- 외부 인터넷 환경에서 접속이 가능해야 하므로, 퍼블릭 아이피로 연결하도록 합니다.

A_MASK	도메인	역할	퍼블릭 아이피
gitlab	pas-mini.io 깃랩 / 도커 레지스트리		52.78.60.43(깃랩)
dcos	pas-mini.io DC/OS 마스터		52.79.125.242(마스터)

config	pas-mini.io 클라우드 콘피그 서버	52.79.51.79(public-agent)
eureka-server	pas-mini.io 유레카 서버	52.79.51.79(public-agent)
iam	pas-mini.io 사용자 인증 서버	52.79.51.79(public-agent)
db	pas-mini.io 데이터베이스	52.79.51.79(public-agent)
cloud-server	pas-mini.io 클라우드 플랫폼 서버	52.79.51.79(public-agent)
cloud	pas-mini.io 클라우드 플랫폼 UI	52.79.51.79(public-agent)

와일드카드(*) A_Mask 를 활용하실 경우, 다음과 같이 간략히 설정이 가능합니다.

A_MASK	도메인	역할	아이피/호스트
gitlab	pas-mini.io	깃랩 / 도커 레지스트리	52.78.60.43(gitlab)
*	pas-mini.io	클라우드 콘피그 서버	52.79.51.79(public-agent)

호스트네임 변경

모든 서버에 각 역할에 맞는 호스트네임으로 변경합니다.

예) bootstrap 서버

— RHEL : /etc/sysconfig/network 파일을 변경합니다.

```
sudo vi /etc/sysconfig/network

# 다음 한줄의 내용으로 교체
HOSTNAME=bootstrap

# 재시작
sudo reboot
```

— CENTOS : /etc/hostname 파일을 변경합니다.

```
sudo vi /etc/hostname

# 다음 한줄의 내용으로 교체
bootstrap

# 적용
sudo hostname bootstrap
```

ssh root 접속 허용

- 원활한 설치를 위해서 모든 서버에 root 권한으로 ssh 접속이 가능하도록 설정합니다.
- 보안 이슈가 있을 경우 설치 후에는 설정을 원복하여도 됩니다.
- 다음 두 파일의 PermitRootLogin 이 no 로 설정되어있을 경우, 주석처리를 하도록 합니다.
 - /etc/ssh/ssh_config
 - /etc/ssh/sshd_config

```
sudo vi /etc/ssh/ssh_config
.
.
# PermitRootLogin no
.
.
sudo service sshd restart
```

네트워크 고정 아이피(옵셔널)

- 네트워크 전용선 과 허브를 구비한 사업장은 이 섹션을 넘어가도 됩니다.
- 퍼블릭 이아스 서비스(AWS 등)을 사용하는 사업장 또한 이 섹션을 넘어가도 됩니다.

유엔진 클라우드 플랫폼은 설치시 지정된 서버들의 프라이빗 아이피가 변경되어서는 안됩니다.
이유는 마스터 노드간의 통신에 필요한 주키퍼 설정이 아이피로 정해지기 때문인데,
아주 많은 컴포넌트들이 이 설정을 이용하므로 설치 후 프라이빗 아이피를 바꾸는 일은 매우 어렵습니다.

그러므로, 만약 이 플랫폼을 가정 또는 교육용으로 설치하실 경우 다음을 체크하세요.

- 클러스터를 구성하는 서버들을 가상화 프로그램을 이용한다. (버추얼 박스 등)
- 가상화서버(VM) 네트워크를 DHCP 서버(공유기) 에서 할당받는다.
- 가상화서버(VM) 네트워크를 내부적으로 고정아이피를 사용하도록 하는 방법을 모르겠다.

위의 해당사항에 모두 해당될 경우, 가상서버를 재시작하거나 공유기가 재부팅 된 경우에 가상서버에 부여되는 아이피 주소가 바뀌게 됩니다.

이를 막기 위해서, 모든 서버에서 고정아이피를 지정하기 위해 아래의 절차를 따르도록 하세요.

1. 네트워크 스크립트 파일을 찾습니다.

경로 이동

```
# cd /etc/sysconfig/network-scripts
```

하위파일 탐색

```
# ll
```

```
-rw-r--r--. 1 root root 126 4월 3 2017 ifcfg-eth0
-rw-r--r--. 1 root root 254 5월 3 2017 ifcfg-lo
lrwxrwxrwx. 1 root root 24 1월 14 04:16 ifdown -> ../../usr/sbin/ifdown
-rwxr-xr-x. 1 root root 1621 8월 3 07:12 ifdown-Team
-rwxr-xr-x. 1 root root 1556 4월 15 2016 ifdown-TeamPort
-rwxr-xr-x. 1 root root 654 5월 3 2017 ifdown-bnep
-rwxr-xr-x. 1 root root 6571 5월 3 2017 ifdown-eth
-rwxr-xr-x. 1 root root 6190 8월 4 10:21 ifdown-ib
-rwxr-xr-x. 1 root root 781 5월 3 2017 ifdown-ipp
-rwxr-xr-x. 1 root root 4540 5월 3 2017 ifdown-ipv6
lrwxrwxrwx. 1 root root 11 1월 14 04:16 ifdown-isdn -> ifdown-ipp
-rwxr-xr-x. 1 root root 1768 5월 3 2017 ifdown-post
-rwxr-xr-x. 1 root root 1068 5월 3 2017 ifdown-ppp
-rwxr-xr-x. 1 root root 870 5월 3 2017 ifdown-routes
-rwxr-xr-x. 1 root root 1456 5월 3 2017 ifdown-sit
-rwxr-xr-x. 1 root root 1462 5월 3 2017 ifdown-tunnel
lrwxrwxrwx. 1 root root 22 1월 14 04:16 ifup -> ../../usr/sbin/ifup
-rwxr-xr-x. 1 root root 1755 4월 15 2016 ifup-Team
-rwxr-xr-x. 1 root root 1876 4월 15 2016 ifup-TeamPort
-rwxr-xr-x. 1 root root 12312 5월 3 2017 ifup-aliases
-rwxr-xr-x. 1 root root 910 5월 3 2017 ifup-bnep
-rwxr-xr-x. 1 root root 12680 5월 3 2017 ifup-eth
-rwxr-xr-x. 1 root root 10114 8월 4 10:21 ifup-ib
-rwxr-xr-x. 1 root root 12075 5월 3 2017 ifup-ipp
-rwxr-xr-x. 1 root root 11893 5월 3 2017 ifup-ipv6
lrwxrwxrwx. 1 root root 9 1월 14 04:16 ifup-isdn -> ifup-ipp
-rwxr-xr-x. 1 root root 650 5월 3 2017 ifup-plip
-rwxr-xr-x. 1 root root 1064 5월 3 2017 ifup-plusb
-rwxr-xr-x. 1 root root 3433 5월 3 2017 ifup-post
-rwxr-xr-x. 1 root root 4154 5월 3 2017 ifup-ppp
-rwxr-xr-x. 1 root root 2001 5월 3 2017 ifup-routes
-rwxr-xr-x. 1 root root 3303 5월 3 2017 ifup-sit
-rwxr-xr-x. 1 root root 2711 5월 3 2017 ifup-tunnel
-rwxr-xr-x. 1 root root 1836 5월 3 2017 ifup-wireless
-rwxr-xr-x. 1 root root 5419 5월 3 2017 init.ipv6-global
-rw-r--r--. 1 root root 18919 5월 3 2017 network-functions
-rw-r--r--. 1 root root 31027 5월 3 2017 network-functions-ipv6
```

2. 네트워크 스크립트 파일 수정

대부분 eth0 이나 eth1 이 메인 네트워크 장비로 잡혀있습니다. ifcfg-eth0 또는 ifcfg-eth1 스크립트를 수정하도록 합니다.

- IPADDR: 설정할 고정아이피
- GATEWAY: 참조-게이트웨이 아이피 보기
- DNS1,...: 참조-DNS 설정 보기

```
#vi ifcfg-eth0

TYPE=Ethernet
BOOTPROTO=static
DEVICE=eth0
ONBOOT=yes
IPADDR=192.168.0.8
NETMASK=255.255.255.0
GATEWAY=192.168.0.1
DNS1=121.88.255.50
DNS2=121.88.255.49

수정 후 네트워크 재시작
# service network restart
```

- 참조: 게이트웨이 아이피 보기

```
ip route | grep default
```

- 참조: DNS 설정 보기

```
nmcli dev show eth0

GENERAL.장치:                eth0
GENERAL.유형:                ethernet
GENERAL.하드웨어주소:      52:54:00:A9:9B:21
GENERAL.MTU:                1500
GENERAL.상태:                100 (연결됨)
GENERAL.연결 :               System eth0
GENERAL.CON-경로:            /org/freedesktop/NetworkManager/ActiveConnection/1
WIRED-PROPERTIES.캐리어:     켜짐
IP4.주소[1]:                192.168.0.8/24
IP4.게이트웨이:              192.168.0.1
IP4.DNS[1]:                  121.88.255.50
IP4.DNS[2]:                  121.88.255.49
IP6.주소[1]:                fe80::5054:ff:fea9:9b21/64
IP6.게이트웨이:              --
```

디스크 마운트(옵셔널)

로그 디렉토리 마운트

- 마스터 서버는 많은 양의 로그가 발생하게 되므로, /var/log 에 충분한 디스크 공간을 할당하길 바랍니다.
- 슬레이브 노드는 도커가 실행되게 되므로, /var/lib/docker 에 충분한 디스크 공간을 할당하길 바랍니다.

디스크 마운트

- OS 가 이미 충분한 양의 디스크에서 운용이 되고있다면 이 단계는 넘어가도 됩니다.
- 하지만 100GB 이하의 디스크 볼륨에서 OS 가 운용되고있거나, 장기적인 운용을 생각한다면 각 서버의 데이터 디렉토리에 대해 별도의 볼륨 디스크를 활용하시길 권장합니다.

모든 서버에 100GB 이상의 볼륨을 생성했다고 가정합니다. 생성한 볼륨을 서버별로 다음의 디렉토리에 마운트를 수행하여야 합니다.

- Master : /var/log
- Slave : /var/lib/docker

다음 명령어로 마운트를 수행할 수 있습니다.

```
$ sudo lsblk
.
.
NAME      MAJ:MIN RM  SIZE RO TYPE MOUNTPOINT
xvda      202:0    0    8G  0 disk
└─xvda1   202:1    0    8G  0 part /
xvdf      202:80   0  100G  0 disk
```

마지막의 xvdf 볼륨이 100G 인 것을 확인할 수 있습니다.

만일 가상서버에 디스크 할당을 하였음에도 볼륨을 확인할 수 없을 경우는 첨부: 파티션 확인, 파티션 추가를 참조하세요.

해당 볼륨을 ext4 로 포맷합니다.

```
$ sudo file -s /dev/xvdf
$ sudo mkfs -t ext4 /dev/xvdf
```

```
.
.
Done
```

1) 에서 정의한 마운트포인트로 마운트합니다.

```
$ sudo mount /dev/xvdf <mount point>
```

예시:

```
$ sudo mount /dev/xvdf /var/lib/docker
```

df 명령어를 통해 마운트 된 사항을 볼 수 있습니다.

```
$ sudo df
.
.
Filesystem      1K-blocks      Used Available Use% Mounted on
/dev/xvda1       8115168 2189708  5490184  29% /
none              4          0          4  0% /sys/fs/cgroup
udev            2009928      12  2009916   1% /dev
tmpfs            404688      404   404284   1% /run
none              5120         0     5120   0% /run/lock
none            2023436      316  2023120   1% /run/shm
none            102400         0   102400   0% /run/user
/dev/xvdf       103081248 448452  97373532   1% /var/lib/docker
```

- 해당 마운트를 시스템 부팅때 자동으로 수행하도록 설정합니다.


```

/etc/fstab 파일을 백업합니다.
$ sudo cp /etc/fstab /etc/fstab.orig

$ sudo vi /etc/fstab
.
.
/dev/xvdf <mount point> ext4 defaults,nofail 0 2
⇒ 해당 라인을 추가합니다.
.
.
$ sudo mount -a

```

- 첨부: 파티션 확인

```

# lsblk

sda                8:0    0   20G    0 disk
├─sda1             8:1    0   500M    0 part /boot
├─sda2             8:2    0   9.5G    0 part
│ └─centos-root    253:0   0   8.5G    0 lvm  /
│ └─centos-swap    253:1   0    1G    0 lvm  [SWAP]
sr0               11:0    1   366K    0 rom

```

- 첨부: 파티션 추가

sda 하위에 파티션을 추가할 경우

```
# sudo su
# fdisk /dev/sda

Welcome to fdisk (util-linux 2.23.2).

Changes will remain in memory only, until you decide to write them.
Be careful before using the write command.


Command (m for help): n
Partition type:
   p   primary (2 primary, 0 extended, 1 free)
   e   extended
Select (default p): p

Select (default 3,4): 3
Selected partition 3

First sector (39845888-41943039, default 39845888):
Using default value 39845888
Last sector, +sectors or +size{K,M,G} (39845888-41943039, default 41943039):
+30G
Partition 3 of type Linux and of size 30G is set

Command (m for help): w

# reboot

# lsblk
```

유틸리티 설치

설치파일 다운로드 && 유틸리티 준비

하기의 모든 사항은, bootstrap 노드에서 진행하도록 합니다.

- 서버에 git, wget 이 있는지 확인하고, 없다면 다음 명령어로 설치합니다.

```
sudo yum install git wget -y
```

Java 설치

[오라클 자바 다운로드 \(http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html\)](http://www.oracle.com/technetwork/java/javase/downloads/jdk8-downloads-2133151.html) 페이지로 이동합니다.

Accept License Agreement 를 클릭하신 후, jdk-8u161-linux-x64.tar.gz 의 링크를 복사합니다.

wget 명령어로 다운받은 후, /opt/ 폴더로 이동시킵니다.

```
# wget --header "Cookie: oraclelicense=accept-securebackup-cookie" <link>
# sudo mv jdk-8u161-linux-x64.tar.gz /opt/
```

이후 설치를 진행합니다.

```
# sudo su
# cd /opt/
# tar xzf jdk-8u161-linux-x64.tar.gz

# cd /opt/jdk1.8.0_161/
# alternatives --install /usr/bin/java java /opt/jdk1.8.0_161/bin/java 2
# alternatives --config java

2 개의 프로그램이 'java'를 제공합니다.

선택      명령
-----
*+ 1      java-1.8.0-openjdk.x86_64 (/usr/lib/jvm/java-1.8.0-openjdk-
1.8.0.151-5.b12.el7_4.x86_64/jre/bin/java)
    2      /opt/jdk1.8.0_161/bin/java

현재 선택[+]을 유지하려면 엔터키를 누르고, 아니면 선택 번호를 입력하십시오:2

alternatives --install /usr/bin/jar jar /opt/jdk1.8.0_161/bin/jar 2
alternatives --install /usr/bin/javac javac /opt/jdk1.8.0_161/bin/javac 2
alternatives --set jar /opt/jdk1.8.0_161/bin/jar
alternatives --set javac /opt/jdk1.8.0_161/bin/javac

# vi /etc/profile

파일의 제일 하단 부분에 다음을 추가합니다.

export JAVA_HOME=/opt/jdk1.8.0_161
export JRE_HOME=/opt/jdk1.8.0_161/jre
export PATH=$PATH:/opt/jdk1.8.0_161/bin:/opt/jdk1.8.0_161/jre/bin

# source /etc/profile
```

Maven 설치

```
# sudo su
# wget http://apache.mirror.cdnetworks.com/maven/maven-3/3.3.9/binaries/apache-
maven-3.3.9-bin.tar.gz
# tar xvf apache-maven-3.3.9-bin.tar.gz
# mv apache-maven-3.3.9 /usr/local/apache-maven

# vi /etc/profile

파일의 제일 하단 부분에 다음을 추가합니다.

export M2_HOME=/usr/local/apache-maven
export M2=$M2_HOME/bin
export PATH=$M2:$PATH

# source /etc/profile
# mvn -version
```

Node 설치

```
# sudo su
# curl --silent --location https://rpm.nodesource.com/setup_7.x | bash -
# yum install nodejs -y

# npm install npm@latest -g
# node -v
# npm -v

# (For phantomJs install)
# yum install bzip2 -y
```

ansible 설치

```
sudo yum install epel-release -y
sudo yum install ansible -y
```

클러스터 설치

설치파일 다운로드

하기 진행순서부터는 root 계정이 아닌, 리눅스 사용자 계정으로 진행하시길 바랍니다 (ex. centos)

설치에 필요한 파일을 다운로드 받습니다.

```
$ cd

# 유엔진 클라우드 다운로드
$ git clone https://github.com/TheOpenCloudEngine/uEngine-cloud

# DCOS 인스톨 파일 다운로드
$ wget https://s3.ap-northeast-2.amazonaws.com/uengine-cloud/dcos_generate_config.sh
```

설정 파일 편집

유엔진 클라우드는 앤서블 유틸리티를 사용해 설치자동화를 지원합니다.

모든 설정이 uEngine-cloud/uengine-resource/config.yml 파일에서 이루어지는데, 이 설정들은 다음과 같은 작업을 자동화하는데 사용되어집니다.

- 네트워크 환경 및 유틸리티 배포
- DC/OS 클러스터 설치
- 클라우드 플랫폼 빌드 및 배포

아래는, 설정 파일의 각 파트별 설명입니다.

SSH and Resolve.conf part

cluster_name	클러스터 이름
ansible_user	ssh 접속 유저
ansible_ssh_private_key_file	ssh 접속 키파일 위치
ssh_port	ssh 포트
resolvers	네임서버 리스트
dns_search	DNS 서치 서버

예제)

```
# SSH and Resolve.conf part
cluster_name: uEngine
ansible_user: centos
ansible_ssh_private_key_file: /home/centos/belugarKey.pem
ssh_port: 22
resolvers:
- 172.31.0.2
dns_search: ap-northeast-2.compute.internal
```

- 주의: ansible_ssh_private_key_file 에 명시된 키파일은 모든 서버에 ssh 접속이 가능한 키여야 합니다.
- 주의: 키파일의 퍼미션이 400 인지 확인하십시오.

여기서 resolvers 와 dns_search 는 다소 생소할 수 있지만, /etc/resolv.conf 조희를 통해 간단히 알아낼 수 있습니다.

```
# cat /etc/resolv.conf

search ap-northeast-2.compute.internal
nameserver 172.31.0.2
```

- nameserver 개수만큼 resolvers 에 넣습니다.
- search 가 있다면 dns_search 에 넣고, 없다면 공란으로 비워둡니다.

Server list part

[서버 준비 \(./pre-server.md\)](#) 문서에서 준비했던 서버들의 아이피를 기입하는 파트입니다.

예)

역할 / 호스트네임	IP 주소	퍼블릭 IP 주소	외부 포트바인딩
bootstrap	172.31.8.143		
master1	172.31.12.143	52.79.125.242	80
master2	172.31.4.125		
master3	172.31.1.198		
public-agent	172.31.5.136	52.79.51.79	80,443,9000 – 60000
agent1	172.31.6.35		
agent2	172.31.1.235		
agent3	172.31.5.245		
agent4	172.31.14.247		
agent5	172.31.7.160		
agent6	172.31.11.70		
agent7	172.31.0.164		
gitlab	172.31.15.249	52.78.60.43	80,5000
ci	172.31.3.61		

설정에제)

```
# Server list part
server:
  private:
    bootstrap: 172.31.8.143
    gitlab: 172.31.15.249
    ci: 172.31.3.61
    public: 172.31.5.136

  master:
    master1: 172.31.12.143
    master2: 172.31.4.125
    master3: 172.31.1.198

  agent:
    agent1: 172.31.6.35
    agent2: 172.31.1.235
    agent3: 172.31.5.245
    agent4: 172.31.14.247
    agent5: 172.31.7.160
    agent6: 172.31.11.70
    agent7: 172.31.0.164

  add-agent:
  gracefully-remove-agent:
  uninstall:

  public:
    # Choice one of your master's public ip
    master: 52.79.125.242
    # Your public node's public ip
    public: 52.79.51.79
```

DB part

유엔진 클라우드 패키지는 대부분의 데이터가 깃랩에 저장되고, 데이터베이스는 아주 적은 양의 데이터만 들어갑니다.

여기에는 사용자 인증정보와 앱 이력 정보가 들어가게 되는데, 데이터베이스가 중단되어도(심지어 데이터를 모두 분실해도) 플랫폼 ui 로 일시적으로 접속을 못할 뿐, 운영중인 앱에 크리티컬한 영향은 주지 않습니다.

이러한 이유로 본 문서에서는 DC/OS 클러스터의 에이전트 노드 중 하나에 Mysql 도커를 실행하는 가이드를 제공합니다. (아래 설정 시 자동으로 수행됩니다.)

- db.ip : private agent 노드 중 하나를 선택하세요. 해당 노드의 /mysql/datadir 에 데이터가 저장됩니다.
- database : 데이터베이스 명
- password : 루트 패스워드

```
# DB part
db:
  ip: 172.31.6.35
  database: uengine
  password: my-secret-pw
```

Domain part

[도메인 준비 \(./pre-domain.md\)](#) 문서에서 준비했던 도메인을 기입하는 파트입니다.

예)

A_MASK	도메인	역할	퍼블릭 아이피
gitlab	pas-mini.io	깃랩 / 도커 레지스트리	52.78.60.43(깃랩)
config	pas-mini.io	클라우드 콘피그 서버	52.79.51.79(public-agent)
eureka-server	pas-mini.io	유레카 서버	52.79.51.79(public-agent)
iam	pas-mini.io	사용자 인증 서버	52.79.51.79(public-agent)
db	pas-mini.io	데이터베이스	52.79.51.79(public-agent)
cloud-server	pas-mini.io	클라우드 플랫폼 서버	52.79.51.79(public-agent)
cloud	pas-mini.io	클라우드 플랫폼 UI	52.79.51.79(public-agent)

설정예제)

```
host:
  registry:
    package: sppark
    private: gitlab.pas-mini.io:5000
    public: gitlab.pas-mini.io:5000
  db: db.pas-mini.io
  gitlab: gitlab.pas-mini.io
  iam: iam.pas-mini.io
  config: config.pas-mini.io
  eureka-server: eureka-server.pas-mini.io
  cloud-server: cloud-server.pas-mini.io
  cloud-ui: cloud.pas-mini.io
```

host.registry?

여기서 registry 란 도커 이미지가 저장되는 이미지 저장소입니다.

registry 서버는 깃랩서버와 동일한 곳에서 실행됨으로, registry.package 와 registry.public 모두 깃랩 도메인을 기입하면 됩니다.

단, 5000 포트를 통해 서비스 될 것임으로 :5000 을 추가해주도록 합니다.

host.registry.package 는 유엔진 클라우드 패키지를 도커 파일을 저장하는 곳인데, 개인 또는 회사의 DockerHub 가 있을 경우 DockerHub 의 아이디를 기입하도록 합니다.(추천)

만일 깃랩 서버에 설치된 registry 서버를 똑같이 사용하고가 하실 경우 마찬가지로 깃랩 도메인:5000 을 기입하도록 합니다.

Security part

사용자 인증정보 서버에 관한 설정입니다.


```
iam:
  port: 80
  client-key: my-client-key
  client-secret: my-client-secret
  admin:
    username: admin
    password: admin
  access-token-lifetime: 7200
  mail:
    host: smtp.gmail.com
    username: sppark@uengine.org
    password: *****
    port: 587
    smtp-auth: true
    smtp-starttls-enable: true
    from-address: sppark@uengine.org
    from-name: uengine
```

- access-token-lifetime : 클라우드 플랫폼 로그인 유지시간입니다.
- admin.username : 인증서버 포탈 접속아이디
- admin.password : 인증서버 포탈 접속패스워드
- mail.**** : 이메일 서버 설정으로, 예제에서는 gmail 을 기준으로 작성되었습니다.

Cloud package part

클라우드 패키지 파트는 [클러스터 설치 \(install-cluster.md\)](#) 와 [깃랩 && 도커 레지스트리 설치 \(install-gitlab.md\)](#) 설치가 종료된 후, 클러스터 정보 및 깃랩 정보를 기입하는 곳입니다.

이 곳의 설정에 관해서는 [클라우드 패키지 빌드 \(install-package.md\)](#) 에서 진행하게 됩니다.

```
# Cloud package part
# Fill out those properties after install DC/OS cluster && Gitlab
# Then, you should re-command "mvn clean install exec:java package"
dcos:
  token:

gitlab:
  root:
    username:
    password:
    token:
  config-repo:
    projectId:
    deployment-path: /deployment
    template-path: /template
```

Summary

전체 설정 파일의 리뷰입니다.

```
# SSH and Resolve.conf part
cluster_name: uEngine
ansible_user: centos
```

```
ansible_ssh_private_key_file: /home/centos/belugarKey.pem
ssh_port: 22
resolvers:
- 172.31.0.2
dns_search: ap-northeast-2.compute.internal

# Server list part
server:
  private:
    bootstrap: 172.31.8.143
    gitlab: 172.31.15.249
    ci: 172.31.3.61
    public: 172.31.5.136

    master:
      master1: 172.31.12.143
      master2: 172.31.4.125
      master3: 172.31.1.198

    agent:
      agent1: 172.31.6.35
      agent2: 172.31.1.235
      agent3: 172.31.5.245
      agent4: 172.31.14.247
      agent5: 172.31.7.160
      agent6: 172.31.11.70
      agent7: 172.31.0.164

    add-agent:
    gracefully-remove-agent:
    uninstall:

  public:
    # Choice one of your master's public ip
    master: 52.79.125.242
    # Your public node's public ip
    public: 52.79.51.79

# DB part
db:
  ip: 172.31.6.35
  database: uengine
  password: my-secret-pw

# Domain part
host:
  registry:
    package: sppark
    private: gitlab.pas-mini.io:5000
    public: gitlab.pas-mini.io:5000
  db: db.pas-mini.io
  gitlab: gitlab.pas-mini.io
```

```

iam: iam.pas-mini.io
config: config.pas-mini.io
eureka-server: eureka-server.pas-mini.io
cloud-server: cloud-server.pas-mini.io
cloud-ui: cloud.pas-mini.io

# Security(IAM) part
iam:
  port: 80
  client-key: my-client-key
  client-secret: my-client-secret
  admin:
    username: admin
    password: admin
  access-token-lifetime: 7200
  mail:
    host: smtp.gmail.com
    username: sppark@uengine.org
    password: *****
    port: 587
    smtp-auth: true
    smtp-starttls-enable: true
    from-address: sppark@uengine.org
    from-name: uengine

# Cloud package part
# Fill out those properties after install DC/OS cluster && Gitlab
# Then, you should re-command "mvn clean install exec:java package"
dcos:
  token:

gitlab:
  root:
    username:
    password:
    token:
  config-repo:
    projectId:
    deployment-path: /deployment
    template-path: /template

```

설정 파일 빌드

uEngine-cloud/uengine-resource/config.yml 을 모두 작성 후, 빌드를 실행합니다.

```

$ cd uengine-resource
$ mvn clean install exec:java package

```

빌드가 성공적으로 진행된다면, uEngine-cloud 폴더에 디렉토리들이 추가로 생성되어, 아래와 같은 모습이 됩니다.

-rw-rw-r--.	1	centos	centos	1074	1월 17	16:53	LICENSE
-rw-rw-r--.	1	centos	centos	22962	1월 17	16:53	README.md
drwxrwxr-x.	5	centos	centos	132	1월 17	17:01	cloud-config-repository
drwxrwxr-x.	2	centos	centos	185	1월 17	17:01	deploys
drwxrwxr-x.	2	centos	centos	145	1월 17	16:53	document
drwxrwxr-x.	3	centos	centos	4096	1월 17	17:09	install
-rwxrwxr-x.	1	centos	centos	2463	1월 17	16:53	pom.xml
drwxrwxr-x.	5	centos	centos	140	1월 17	16:53	template-iam
drwxrwxr-x.	4	centos	centos	123	1월 17	16:53	template-springboot
drwxrwxr-x.	8	centos	centos	4096	1월 17	16:53	template-vuejs
drwxrwxr-x.	4	centos	centos	123	1월 17	16:53	template-zuul
drwxrwxr-x.	3	centos	centos	50	1월 17	16:53	uengine-cloud-config
drwxrwxr-x.	5	centos	centos	118	1월 17	16:53	uengine-cloud-iam
drwxrwxr-x.	3	centos	centos	50	1월 17	16:53	uengine-cloud-server
drwxrwxr-x.	7	centos	centos	259	1월 17	16:53	uengine-cloud-ui
drwxrwxr-x.	3	centos	centos	50	1월 17	16:53	uengine-eureka-server
drwxrwxr-x.	8	centos	centos	230	1월 17	17:01	uengine-resource

DC/OS 클러스터 설치

유틸리티 및 도커 프로비저닝

```
$ cd install
$ sudo sh -c "cat ansible-hosts.yml > /etc/ansible/hosts"

$ ansible-playbook ansible-install.yml

PLAY [all]
*****
*****

TASK [Gathering Facts]
*****
*****
ok: [172.31.3.61]
ok: [172.31.8.143]
ok: [172.31.1.235]
ok: [172.31.6.35]
.
.
```

- 팁: 단계별로 실행을 원할 경우 --step 옵션을 추가하도록 합니다.

```
$ ansible-playbook --step ansible-install.yml
```

DC/OS 클러스터 인스톨

배포 패키지 생성

```
# copy your ssh key file to genconf folder
$ cp <your-private-key-file-path> ./genconf/ssh_key

$ mv ~/dcos_generate_config.sh ./
$ sudo bash dcos_generate_config.sh --genconf

====> EXECUTING CONFIGURATION GENERATION
Generating configuration files...
Package filename: packages/dcos-config/dcos-config--
setup_5a900644a2b78900d7420b5d904c19a7e24d539b.tar.xz
Package filename: packages/dcos-metadata/dcos-metadata--
setup_5a900644a2b78900d7420b5d904c19a7e24d539b.tar.xz
Generating Bash configuration files for DC/OS
```

preflight 는 사전 체크 단계입니다. 이 단계에서 모든 사항에 대해 pass 가 나오지 않는다면, [트러블 슈팅 \(trouble.md\)](#) 을 통해 해결하세요.

```
sudo bash dcos_generate_config.sh --preflight
```

```
====> EXECUTING_PREFLIGHT
====> START run_preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> STAGE preflight_cleanup
====> OUTPUT FOR run_preflight
.
.
```

deploy 는 설치 단계입니다. 수분 이상 소요될 수 있습니다.

```
sudo bash dcos_generate_config.sh --deploy
```

postflight 는 설치 후 확인 단계입니다. 이 단계에서 모든 DC/OS 컴포넌트의 헬스체크를 하게 되며, 만약 이 단계에서 수분간 동작을 멈추거나 Fail 이 떨어지게 된다면, 트러블 슈팅을 통해 이슈를 해결하셔야 합니다.

```
sudo bash dcos_generate_config.sh --postflight
```

클러스터 상태 점검

설치 수분 후, 정상적으로 설치가 되었다면 마스터 노드 중 한 곳에 접속하였을 때, 아래의 포트 리스트와 동일한 포트가 존재해야 합니다.

```
ssh -i <your-key-file> master1
```

(No info could be read for "-p": geteuid()=1000 but you should be root.)

Active Internet connections (only servers)

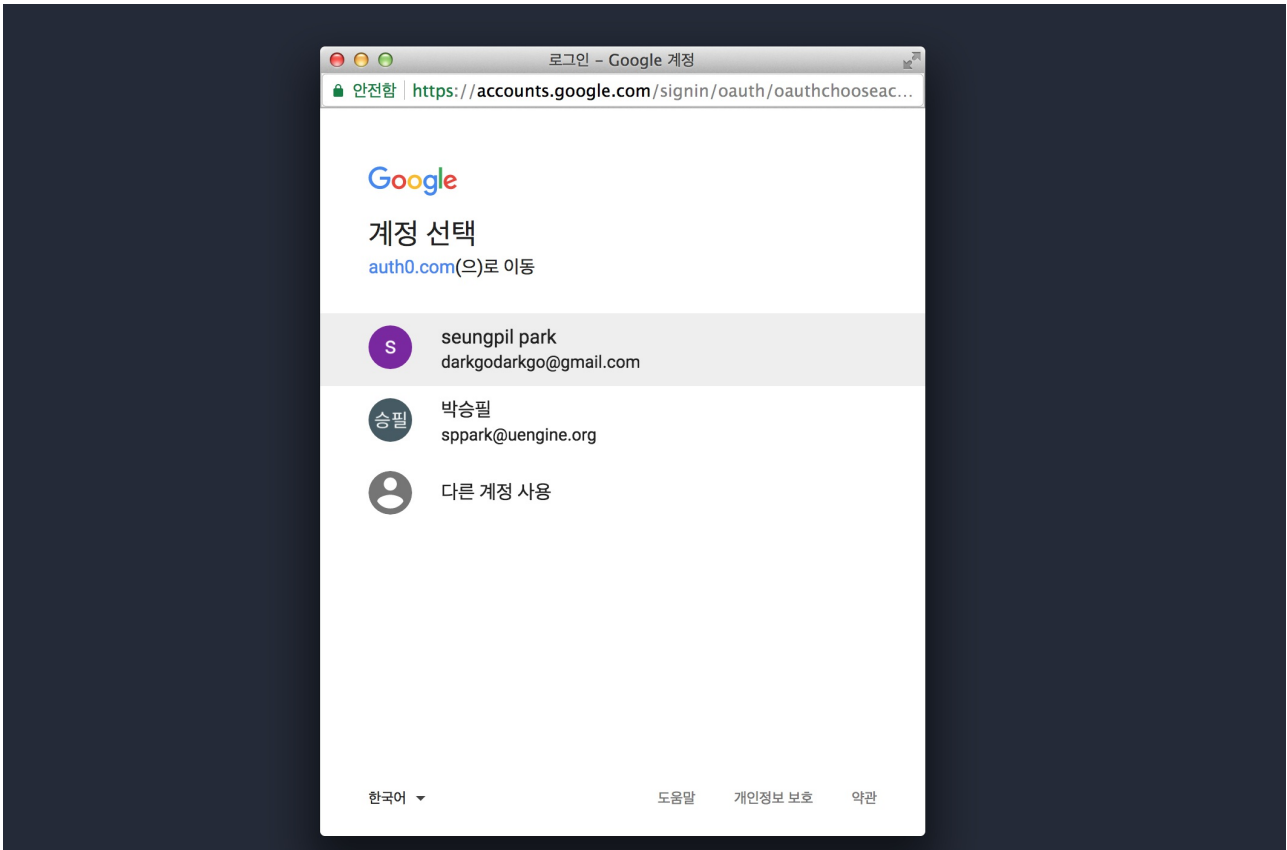
Proto	Recv-Q	Send-Q	Local Address	Foreign Address	State
PID/Program name					
tcp	0	0	0.0.0.0:61420	0.0.0.0:*	LISTEN
-					
tcp	0	0	198.51.100.3:63053	0.0.0.0:*	LISTEN
-					
tcp	0	0	198.51.100.2:63053	0.0.0.0:*	LISTEN
-					
tcp	0	0	198.51.100.1:63053	0.0.0.0:*	LISTEN
-					
tcp	0	0	172.31.12.143:63053	0.0.0.0:*	LISTEN
-					
tcp	0	0	172.17.0.1:63053	0.0.0.0:*	LISTEN
-					
tcp	0	0	127.0.0.1:63053	0.0.0.0:*	LISTEN
-					
tcp	0	0	127.0.0.1:15055	0.0.0.0:*	LISTEN
-					
tcp	0	0	0.0.0.0:111	0.0.0.0:*	LISTEN
-					
tcp	0	0	0.0.0.0:80	0.0.0.0:*	LISTEN
-					
tcp	0	0	198.51.100.3:53	0.0.0.0:*	LISTEN
-					
tcp	0	0	198.51.100.2:53	0.0.0.0:*	LISTEN
-					
tcp	0	0	198.51.100.1:53	0.0.0.0:*	LISTEN
-					
tcp	0	0	172.31.12.143:53	0.0.0.0:*	LISTEN
-					
tcp	0	0	172.17.0.1:53	0.0.0.0:*	LISTEN
-					
tcp	0	0	127.0.0.1:53	0.0.0.0:*	LISTEN
-					
tcp	0	0	0.0.0.0:22	0.0.0.0:*	LISTEN
-					
tcp	0	0	127.0.0.1:25	0.0.0.0:*	LISTEN
-					
tcp	0	0	172.31.12.143:5050	0.0.0.0:*	LISTEN
-					

tcp	0	0 0.0.0.0:443	0.0.0.0:*	LISTEN
-				
tcp	0	0 127.0.0.1:62080	0.0.0.0:*	LISTEN
-				
tcp	0	0 172.31.12.143:15201	0.0.0.0:*	LISTEN
-				
tcp	0	0 127.0.0.1:62053	0.0.0.0:*	LISTEN
-				
tcp	0	0 0.0.0.0:62501	0.0.0.0:*	LISTEN
-				
tcp	0	0 127.0.0.1:8101	0.0.0.0:*	LISTEN
-				
tcp	0	0 0.0.0.0:62502	0.0.0.0:*	LISTEN
-				
tcp	0	0 0.0.0.0:8008	0.0.0.0:*	LISTEN
-				
tcp6	0	0 :::61420	:::*	LISTEN
-				
tcp6	0	0 :::111	:::*	LISTEN
-				
tcp6	0	0 :::8080	:::*	LISTEN
-				
tcp6	0	0 :::3888	:::*	LISTEN
-				
tcp6	0	0 :::8181	:::*	LISTEN
-				
tcp6	0	0 :::9942	:::*	LISTEN
-				
tcp6	0	0 :::22	:::*	LISTEN
-				
tcp6	0	0 :::41303	:::*	LISTEN
-				
tcp6	0	0 ::1:25	:::*	LISTEN
-				
tcp6	0	0 :::1050	:::*	LISTEN
-				
tcp6	0	0 :::8123	:::*	LISTEN
-				
tcp6	0	0 :::61053	:::*	LISTEN
-				
tcp6	0	0 127.0.0.1:7070	:::*	LISTEN
-				
tcp6	0	0 :::2181	:::*	LISTEN
-				
tcp6	0	0 127.0.0.1:9990	:::*	LISTEN
-				
tcp6	0	0 :::9000	:::*	LISTEN
-				

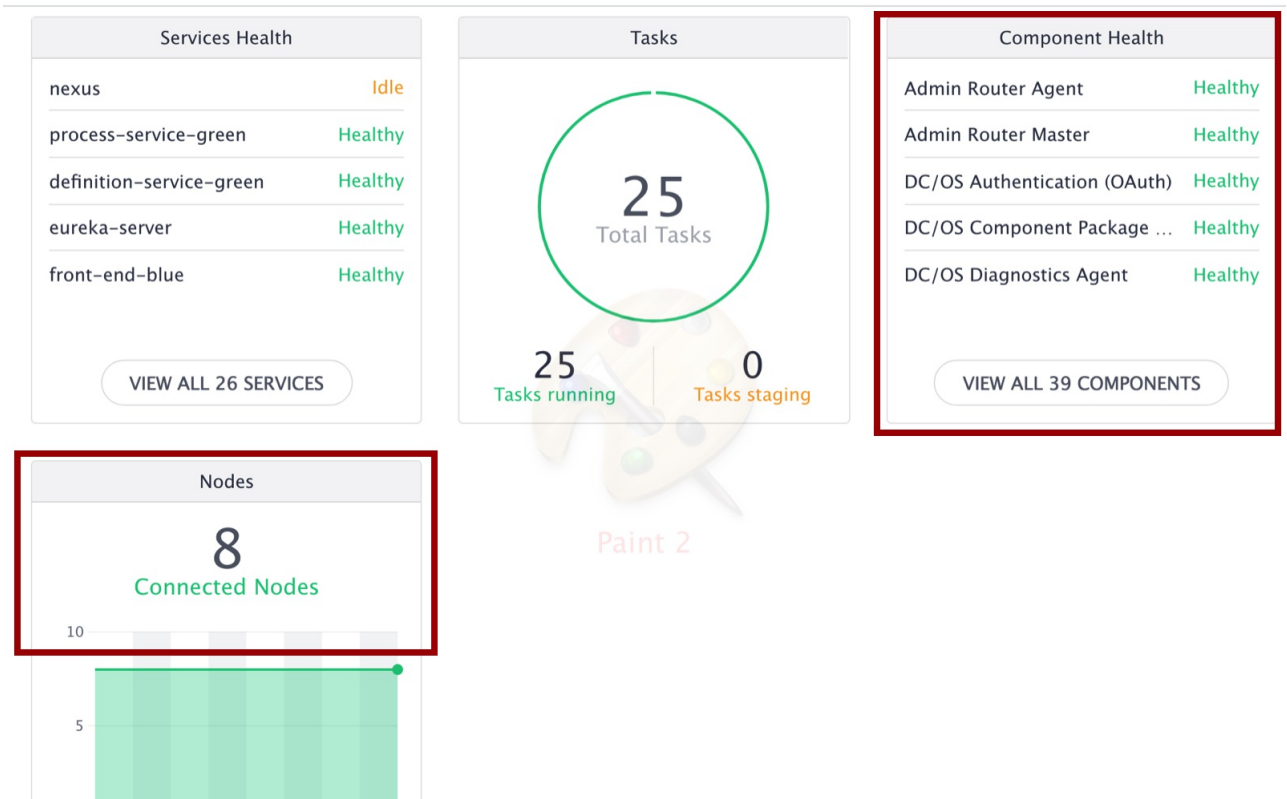
DC/OS 의 웹 UI 를 통해 모든 컴포넌트들이 동작하고 있는지 확인합니다.

마스터 서버중 한 곳을 웹 브라우저를 통해 접속해봅니다. <http://52.79.125.242> (<http://52.79.125.242>)

DC/OS 웹 UI 에 로그인하기 위해서는 구글 계정이 필요합니다.



그림과 같이 모든 컴포넌트와 노드들이 health 상태에 있는지 확인합니다.



문제가 있는 컴포넌트가 있다면 원인분석을 위해 [컴포넌트 헬스 체크 \(trouble-component.md\)](#) 를 참조하기 바랍니다.

잘못된 설정값으로 초기 설치를 진행하여, 다시 설치 시도시에 원활히 동작되지 않는 경우 초기 설치요소를 완전히 제거하고 재설치 할 필요가 있습니다.

[설치 완전 삭제 \(op-uninstall.md\)](#) 를 참조하여 진행하도록 하십시오.

클러스터 계정 및 CLI

사용자 생성

DC/OS 엔터프라이즈에서는 많은 시큐리티 설정 옵션이 있지만, DC/OS 커뮤니티 버전은 OAuth0 클라우드 서비스를 통해서만 웹 UI 및 Rest Api 를 접속합니다. (로그아웃 주기도 매우 짧습니다.)

해당 문제를 해결하기 위해, 아래의 로직에 따라 거의 영구적으로 사용할 수 있는 어세스 토큰을 생성하도록 하겠습니다.

1. 마스터 노드 중 한곳으로 이동 후, 영구적인 토큰을 생성합니다. gmail 계정이 하나 필요합니다.

```
$ ssh -i <your-key-file> master1

# 사용자 생성
$ sudo su
# dcos-shell /opt/mesosphere/bin/dcos_add_user.py darkgodarkgo@gmail.com

# 영구적 토큰 생성 워크어라운드
# export DCOS_USER=darkgodarkgo@gmail.com
# DCOS_ACS_TOKEN="$(docker run --rm -v /var/lib/dcos/dcos-oauth/auth-token-secret:/key karlkfi/jwt-encoder ${DCOS_USER} /key --duration=86400000)"
# echo $DCOS_ACS_TOKEN

# 토큰 테스트
# curl --header "Authorization: token=eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiJkYXJrZ29kYXJrZ29AZ21haWwuY29tIiwiaXhwIjozMTI1NTU5MDY0MzkuODk0MX0.aHgH_M-g-n-WlnPg_CorMGYEprULPSeUTIGu3GyZQ-U" http://localhost/service/marathon/v2/apps
```

획득한 토큰은 깃랩 데이터베이스 설치에 필요하니 기억하고 있도록 합니다.

CLI 설치

부트스트랩 노드로 돌아와서, 다음을 실행합니다.

```
$ cd
$ curl -O https://downloads.dcos.io/binaries/cli/linux/x86-64/dcos-1.10/dcos
$ sudo mv dcos /usr/local/bin
$ chmod +x /usr/local/bin/dcos

# 마스터 노드 중 하나로 cli 를 등록하도록 합니다.
# CLI 클라이언트는 사용자 등록을 위해 웹브라우저 주소를 리다이렉트 합니다. 그러므로, 마스터 노드의 퍼블릭 아이피를 넣어주도록 합니다.
# 네트워크 환경이 프라이빗 아이피로도 마스터 노드로의 브라우저 접속이 가능하다면, 프라이빗 아이피 그대로 사용하셔도 무방합니다.

dcos cluster setup http://52.79.125.242

If your browser didn't open, please go to the following link:

    http://52.79.125.242/login?redirect_uri=urn:ietf:wg:oauth:2.0:oob
Enter OpenID Connect ID Token:
```



콘솔의 url 로 이동하여, 화면에 나오는 토큰값을 커맨드 프롬프트에 입력하면 CLI 를 사용가능 합니다.

깃랩 && 도커 레지스트리 설치

다음 절차는 깃랩 서버에서 진행하게 됩니다.

```
$ ssh -i <your-private-key> gitlab
```

깃랩 서버는 클라우드 플랫폼의 앱 정보, 환경설정 정보, 그리고 소스코드 형상관리에 사용되어집니다.

Omnibus install

깃랩의 설치 방식에는 Omnibus 패키지와 메뉴얼 설치가 있습니다.

Omnibus 패키지 로 설치를 진행하실 경우 깃랩 패키지에 포함된 Chef 프로비저닝 툴에 의해

- Postgres 데이터베이스
- 사이드킥 (큐 서버)
- 메일 서버
- 깃
- 루비 레일스
- 캐쉬 서버

들이 함께 설치되게 됩니다.

Omnibus 패키지는 또한 Daily 백업, 리스토어에 있어 편리한 인터페이스를 제공합니다.

다음의 명령어로 설치를 진행하도록 합니다.

```
$ sudo yum install -y curl policycoreutils-python openssh-server
$ sudo systemctl enable sshd
$ sudo systemctl start sshd
$ sudo firewall-cmd --permanent --add-service=http
$ sudo systemctl reload firewalld

$ curl https://packages.gitlab.com/install/repositories/gitlab/gitlab-
ce/script.rpm.sh | sudo bash

$ sudo yum install -y gitlab-ce-10.2.4-ce.0.el7.x86_64
```

기본 환경 설정

깃랩은 IP 로 접속하는 것 보다, FQDN 형식의 external_url 을 설정하여야 프로젝트 주소 및 Push,Commit Remote url 관리시 편리함이 있습니다.

또한 이메일 서버 정보가 있으면, 자신이 수행한 CI 작업 실패시 알림메일을 받을 수 있습니다.

- /etc/gitlab/gitlab.rb 파일을 수정합니다. 처음에는 모두 주석처리되어있으므로, 파일의 가장 상단에 아래의 내용을 설치환경에 맞게 넣어주면 됩니다.

```
$ sudo vi /etc/gitlab/gitlab.rb

.
.
external_url 'http://gitlab.pas-mini.io'

gitlab_rails['smtp_enable'] = true
gitlab_rails['smtp_address'] = "smtp.gmail.com"
gitlab_rails['smtp_port'] = 587
gitlab_rails['smtp_user_name'] = "flamingo.workflow@gmail.com"
gitlab_rails['smtp_password'] = "princoprinco9"
gitlab_rails['smtp_domain'] = "smtp.gmail.com"
gitlab_rails['smtp_authentication'] = "login"
gitlab_rails['smtp_enable_starttls_auto'] = true
gitlab_rails['smtp_tls'] = false
.
.
```

설정이 끝난 후에는 reconfigure 를 수행하도록 합니다. 이는 Chef 클라이언트가 설정대로 컴포넌트를 재배포하여, 깃랩 Rails 서버를 재구동시킵니다.

```
$ sudo gitlab-ctl reconfigure
```

도커 레지스트리 설치

다음은 도커 레지스트리 (도커 이미지 저장소) 설치과정입니다. 깃랩과 동일한 서버에서 진행하도록 합니다.

도커 레지스트리는 깃랩 CI 에서 생성된 도커이미지를 저장하였다가, 클라우드 플랫폼에서 배포 요청시 이미지를 보내주는 역할을 합니다.

아래 명령어는 항시적으로 registry 서버가 떠있도록 해줍니다.

```
sudo docker run -d -p 5000:5000 --restart=always --name registry registry:2
```

CI 서버 설치

러너 설치

다음은 CI 서버에서 수행하도록 합니다.

깃랩 형상관리 코드로부터 실제 빌드를 수행하려면 Go 로 작성된 **GitLab Runner** 를 설치해야합니다.

.Net, Java, Python, C, PHP 및 기타 프로그래밍 언어를 테스트 할 수 있으며, **GitLab Runner** 에는 오토 스케일, Docker 지원 및 병렬 작업 기능 등 많은 기능이 있습니다.

다음의 명령어로 간단히 설치가 가능합니다.

```
$ curl -L https://packages.gitlab.com/install/repositories/runner/gitlab-runner/script.rpm.sh | sudo bash

$ sudo yum install gitlab-runner -y
```

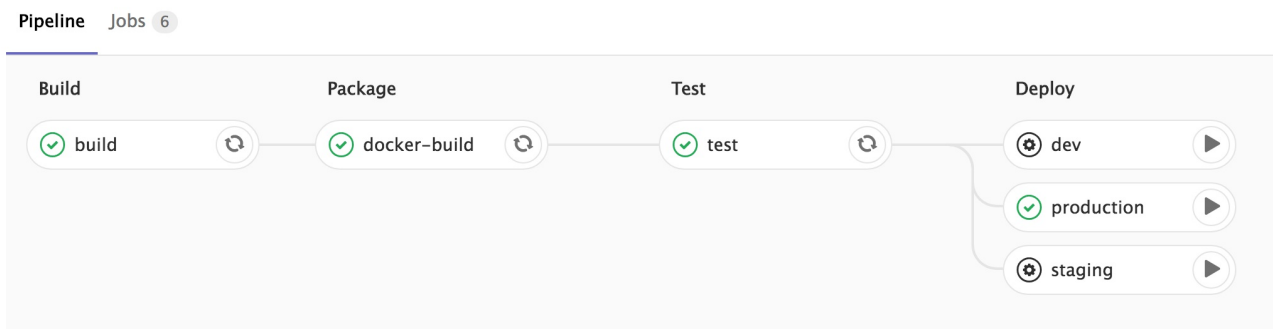
CI 서버를 별도로 준비하는 이유

CI 서버는 고사양일 필요 없이, EC2 기준 t2.micro (1cpu,1GB) 로도 충분합니다.

다만, 동시 다발적인 CI 작업이 이루어질 경우, 많은 양의 CI 캐쉬 데이터와 형상관리 코드가 오가게 됨으로, 원한할 네트워크 환경이 필요합니다.

CI 캐쉬 데이터가 필요한 이유는 매번 빌드때마다 maven, node install 등의 작업은 오랜 다운로드 시간을 소모하게 되는데, 이러한 시간 소모를 막고자 빌드 결과물 디렉토리 ex) m2,node_modules 를 저장할 수 있기 때문입니다.

또한, CI 빌드의 각 스테이지는 독립된 컨테이너에서 실행되게 되는데, 캐쉬 데이터는 스테이지 사이의 빌드 결과물을 가지고 연속된 작업을 가능하게 합니다.



- CI 캐쉬 를 클라우드에 저장하지 않고 로컬에 저장할 경우, 충분한 디스크 스페이스를 확보하는걸 추천합니다.

러너 설정

러너와 깃랩을 연동하기 위해서 깃랩 웹 UI 로 접속합니다.

초기 접속 아이디/비밀번호는 root , adminadmin 입니다.

GitLab Community Edition

Open source software to collaborate on code

Manage Git repositories with fine-grained access controls that keep your code secure. Perform code reviews and enhance collaboration with merge requests. Each project can also have an issue tracker and a wiki.

Sign in	Register
Username or email	
<input type="text" value="root"/>	
Password	
<input type="password" value="....."/>	
<input type="checkbox"/> Remember me	Forgot your password?
<input type="button" value="Sign in"/>	

Didn't receive a confirmation email? [Request a new one.](#)

러너와 깃랩 연동에는 깃랩 서버에서 발급한 토큰이 필요한데, Admin 메뉴의 Overview => Runners 항목에서 볼 수 있습니다.

Admin Area > Runners

A 'Runner' is a process which runs a job. You can setup as many Runners as you need. Runners can be placed on separate users, servers, even on your local machine.

Each Runner can be in one of the following states:

- shared** - Runner runs jobs from all unassigned projects
- specific** - Runner runs jobs from assigned projects
- locked** - Runner cannot be assigned to other projects
- paused** - Runner will not receive any new jobs

You can reset runners registration token by pressing a button below.

How to setup a shared Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the [GitLab Runner section](#) for information on how to install it).
2. Specify the following URL during the Runner setup: <http://gitlab.pas-mini.io/>
3. Use the following registration token during setup: **Kndb7ac3-4__EfEwxB_o**
4. Start the Runner!

위의 그림의 우측 하단 빨간색 박스에 해당하는 것이 러너 등록 토큰입니다.

이 등록토큰을 가지고, CI 서버에서 다음 명령어로 러너를 등록합니다.

```
$ sudo gitlab-ci-multi-runner register -n \  
--url http://gitlab.pas-mini.io\  
--registration-token Kndb7ac3-4__EfEwxB_o \  
--executor docker \  
--description "uengine docker runner:shared-socket" \  
--docker-image "docker:latest" \  
--docker-volumes /var/run/docker.sock:/var/run/docker.sock
```

description 항목에 uengine docker runner:shared-socket 으로 등록하셔야 하며, 클라우드 플랫폼은 이 러너를 사용하여 작업을 수행합니다.

생성후에 /etc/gitlab-runner/config.toml 파일을 수정할 경우 좀 더 상세한 설정이 가능합니다.

```

concurrent = 5
check_interval = 0
[[runners]]
  name = "uengine docker runner:shared-socket"
  url = "http://gitlab.uengine.io"
  token = "306c411ef28850993d24faa3a05204"
  executor = "docker"
  [runners.docker]
    tls_verify = false
    image = "docker:latest"
    privileged = false
    disable_cache = false
    volumes =
  ["/root/m2:/root/.m2", "/var/run/docker.sock:/var/run/docker.sock", "/cache"]
    shm_size = 0
  [runners.cache]

```

`concurrent` 는 이 러너를 통해 동시에 수행할 수 있는 작업 수를 뜻합니다. 이 항목을 넘어서는 CI 요청이 있을 경우, 깃랩 UI 상에서 해당 작업은 pending 상태에 놓이게 됩니다.

`executor` 는 러커가 실행될 환경을 뜻하는데, `docker` 로 설정할 경우 완전히 격리된 컨테이너에서 CI 빌드를 수행하게 됩니다.

`executor` 를 `docker` 로 설정할 경우 `volumes` 또한 마더 머신(깃랩 서버) 의 `docker` 자원과 공유하기 위해 도커 소켓 볼륨을 추가한 모습입니다.

설정이 종료된 후에는 다음 명령어로 러너를 재시작합니다.

```
$ sudo gitlab-ci-multi-runner restart
```

Admin 메뉴의 Overview => Runners 항목에서 다음과 같이 러너가 등록되었다면 절차를 마친 것입니다.

How to setup a shared Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the [GitLab Runner section](#) for information on how to install it).
2. Specify the following URL during the Runner setup: <http://gitlab.pas-mini.io/>
3. Use the following registration token during setup: `Kndb7ac3-4__EfEwxB_o`
4. Start the Runner!

Runner description or to

Search

Runners with last contact more than a minute ago: 1

Type	Runner token	Description	Version	Projects	Jobs	Tags	Last contact	
shared locked	9197e259	uengine docker runner:shared-socket	10.3.0	n/a	600		6 minutes ago	  

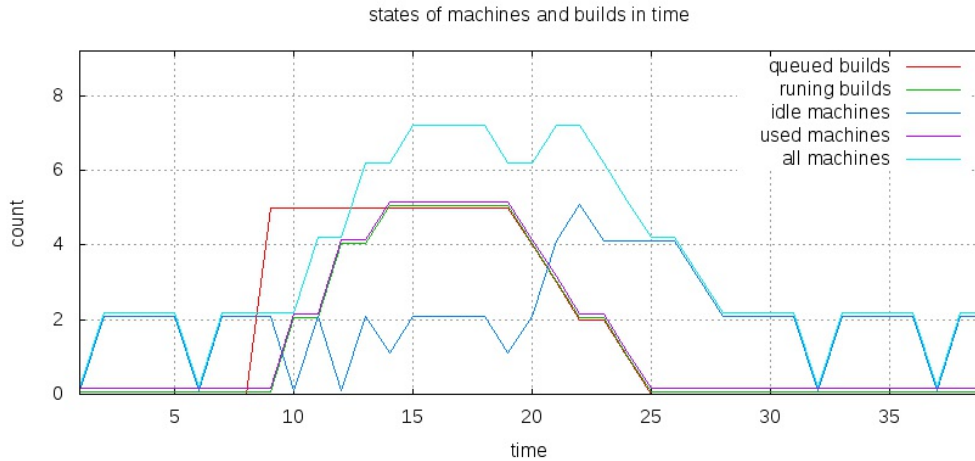
CI 오토스케일 설정

소규모 단일 팀이나 개인에 의해 운영되는 환경은 Gitlab 서버의 리소스 자원이 충분하다면 문제 될 게 없지만, 그보다 큰 빈번한 커밋 작업이 이루어지게 되는 개발환경에서는

CI 작업을 단일 서버에서 수행하기에는 무리가 있습니다.

잘 알려진 jenkins slave 처럼, 슬레이브 노드 풀을 지정해 놓고 CI 작업 요청을 분산시키는 방식 대신, 깃랩 CI 는 Docker machine 을 통해 CI 요청 수에 따라 AWS, DigitalOcean, GoogleCloud, VirtualBox, Openstack 등의 다양한 인프라에 필요한 자원을 오토스케일링 하게 됩니다.

CI 작업 수와 Docker machine 수 의 관계



이 환경에 대한 문서는 [Autoscale GitLab CI runners and save 90% on EC2 costs](https://about.gitlab.com/2017/11/23/autoscale-ci-runners/)
(<https://about.gitlab.com/2017/11/23/autoscale-ci-runners/>)
과 [Runners autoscale configuration](https://docs.gitlab.com/runner/configuration/autoscale.html#distributed-runners-caching)
(<https://docs.gitlab.com/runner/configuration/autoscale.html#distributed-runners-caching>) 에서
더 자세히 볼 수 있습니다.

도커 머신 설치

```
$ curl -L https://github.com/docker/machine/releases/download/v0.12.2/docker-machine-`uname -s`-`uname -m` >/tmp/docker-machine &&  
chmod +x /tmp/docker-machine &&  
sudo cp /tmp/docker-machine /usr/local/bin/docker-machine
```

캐쉬 저장소 등록

CI 는 기본적으로 자신이 설치된 머신의 디렉토리에 캐쉬 데이터 를 저장하게 됩니다. 하지만, 오토 스케일 환경에서는 도커 머신이 분산배치 되기 때문에, 캐쉬 데이터를 주고 받을 수 있는 저장소가 필요합니다.

S3 storage

다음은 AWS S3 스토리지를 사용한 /etc/gitlab-runner/config.toml 설정입니다.

```
[[runners]]
.
.
[runners.cache]
  Type = "s3"
  ServerAddress = "172.31.3.61:9005"
  AccessKey = "AKIAI5U6D*****"
  SecretKey = "uVHmnl0Kz*****"
  BucketName = "uengine-gitlab-runner"
  BucketLocation = "ap-northeast-2"
```

Local cache storage

보안 환경에 따라서 내부에 캐시 데이터를 저장해야 하는 경우, 내부적으로 S3 서비스를 설치하여 사용할 수 있습니다. 참조문서 [Install the cache server \(https://docs.gitlab.com/runner/install/autoscaling.html#install-the-cache-server\)](https://docs.gitlab.com/runner/install/autoscaling.html#install-the-cache-server)

1. Login to a dedicated machine where the cache server will be running
2. Make sure that Docker Engine is installed on that machine
3. Start [minio \(https://www.minio.io/\)](https://www.minio.io/), a simple S3-compatible server written in Go:

```
docker run -it --restart always -p 9005:9000 \
  -v /.minio:/root/.minio -v /export:/export \
  --name minio \
  minio/minio:latest server /export
```

You can modify the port 9005 to expose the cache server on different port.

4. Check the IP address of the server:

```
hostname --ip-address
```

5. Your cache server will be available at MY_CACHE_IP:9005
6. Read the Access and Secret Key of minio with: `sudo cat /.minio/config.json`
7. Create a bucket that will be used by the Runner: `sudo mkdir /export/runner.runner` is the name of the bucket in that case. If you choose a different bucket then it will be different
8. All caches will be stored in the `/export` directory

Configure Local cache runner

로컬 캐시 서버를 구축 한 후에는, `/etc/gitlab-runner/config.toml` 러너환경에 이 서버를 사용하도록 설정합니다.


```
[[runners]]
.
.
  [runners.cache]
    Type = "s3"
    ServerAddress = "MY_CACHE_IP:9005"
    AccessKey = "ACCESS_KEY"
    SecretKey = "SECRET_KEY"
    BucketName = "runner"
    Insecure = true # Use Insecure only when using with Minio, without the
TLS certificate enabled
```

AccessKey 와 SecretKey 는 `sudo cat /.minio/config.json` 에 위치해 있습니다.

러너 오토 스케일 설정

참조문서 [Runners autoscale configuration](#)

(<https://docs.gitlab.com/runner/configuration/autoscale.html#distributed-runners-caching>)

본 예제에서는 EC2 환경의 스팟 인스턴스 를 CI 풀로 사용하도록 `/etc/gitlab-runner/config.toml` 에 설정한 예입니다.

```

concurrent = 12
check_interval = 0
[[runners]]
  name = "uengine docker runner:shared-socket"
  url = "http://172.31.15.249"
  token = "9197e25967da8c39bc0a826f148464"
  executor = "docker+machine"
  limit = 20
[runners.docker]
  tls_verify = false
  image = "docker:latest"
  privileged = false
  disable_cache = false
  volumes = ["/var/run/docker.sock:/var/run/docker.sock", "/cache"]
  shm_size = 0
[runners.machine]
  IdleCount = 1
  MachineDriver = "amazonec2"
  MachineName = "runner-%s"
  MachineOptions = [
    "amazonec2-access-key=AKIAI5U*****",
    "amazonec2-secret-key=uVHmnl0Kz3s0R1*****",
    "amazonec2-region=ap-northeast-2",
    "amazonec2-instance-type=t2.medium",
    "amazonec2-vpc-id=vpc-e8f91b81",
    "amazonec2-subnet-id=subnet-856487ec",
    "engine-insecure-registry=gitlab.pas-mini.io:5000"
  ]
  IdleTime = 1800
[runners.cache]
  Type = "s3"
  ServerAddress = "s3.amazonaws.com"
  AccessKey = "AKIAI5U*****"
  SecretKey = "uVHmnl0Kz3s0R1*****"
  BucketName = "uengine-gitlab-runner"
  BucketLocation = "ap-northeast-2"

```

먼저, executor 항목이 "docker+machine" 으로 설정한 것을 볼 수 있습니다. 이는 CI 서버에 설치된 도커 머신이 다른 기기에 설치된 도커로 명령을 대리 전달하고, 대리 전달받은 기기의 도커가 부팅되면서 깃랩 소스코드, 캐쉬 데이터를 받아와 작업을 수행하게 됩니다.


How to setup a shared Runner for a new project

1. Install a Runner compatible with GitLab CI (checkout the [GitLab Runner section](#) for information on how to install it).
2. Specify the following URL during the Runner setup: <http://gitlab.pas-mini.io/>
3. Use the following registration token during setup: `Kndb7ac3-4__EfEwxB_o`
4. Start the Runner!

Runner description or to

Search

Runners with last contact more than a minute ago: 1

Type	Runner token	Description	Version	Projects	Jobs	Tags	Last contact	
shared locked	9197e259	uengine docker runner:shared-socket	10.3.0	n/a	600		6 minutes ago	  

runners.machine 항목에는 이 도커 머신을 어떻게 띄울 것인가에 대한 기술인데, 도커 머신은 깃랩에 포함된 패키지가 아닌 도커 오피셜 클라이언트입니다.
따라서, 이 항목에는 [공식 도커 머신 \(https://docs.docker.com/machine/\)](https://docs.docker.com/machine/) 을 참조하며 기술하면 됩니다. 지원되는 도커 머신 드라이버 리스트는 다음이 있습니다.

[Amazon Web Services \(https://docs.docker.com/machine/drivers/aws/\)](https://docs.docker.com/machine/drivers/aws/)

[Microsoft Azure \(https://docs.docker.com/machine/drivers/azure/\)](https://docs.docker.com/machine/drivers/azure/)

[Digital Ocean \(https://docs.docker.com/machine/drivers/digital-ocean/\)](https://docs.docker.com/machine/drivers/digital-ocean/)

[Exoscale \(https://docs.docker.com/machine/drivers/exoscale/\)](https://docs.docker.com/machine/drivers/exoscale/)

[Google Compute Engine \(https://docs.docker.com/machine/drivers/gce/\)](https://docs.docker.com/machine/drivers/gce/)

[Generic \(https://docs.docker.com/machine/drivers/generic/\)](https://docs.docker.com/machine/drivers/generic/)

[Microsoft Hyper-V \(https://docs.docker.com/machine/drivers/hyper-v/\)](https://docs.docker.com/machine/drivers/hyper-v/)

[OpenStack \(https://docs.docker.com/machine/drivers/openstack/\)](https://docs.docker.com/machine/drivers/openstack/)

[Rackspace \(https://docs.docker.com/machine/drivers/rackspace/\)](https://docs.docker.com/machine/drivers/rackspace/)

[IBM Softlayer \(https://docs.docker.com/machine/drivers/soft-layer/\)](https://docs.docker.com/machine/drivers/soft-layer/)

[Oracle VirtualBox \(https://docs.docker.com/machine/drivers/virtualbox/\)](https://docs.docker.com/machine/drivers/virtualbox/)

[VMware vCloud Air \(https://docs.docker.com/machine/drivers/vm-cloud/\)](https://docs.docker.com/machine/drivers/vm-cloud/)

[VMware Fusion \(https://docs.docker.com/machine/drivers/vm-fusion/\)](https://docs.docker.com/machine/drivers/vm-fusion/)

[VMware vSphere \(https://docs.docker.com/machine/drivers/vsphere/\)](https://docs.docker.com/machine/drivers/vsphere/)

[VMware Workstation \(https://github.com/pecigonzalo/docker-machine-vmwareworkstation\)](https://github.com/pecigonzalo/docker-machine-vmwareworkstation)

(unofficial plugin, not supported by Docker)

[Grid 5000 \(https://github.com/Spirals-Team/docker-machine-driver-g5k\)](https://github.com/Spirals-Team/docker-machine-driver-g5k) (unofficial plugin, not supported by Docker)

설정이 종료된 후에는 다음 명령어로 러너를 재시작합니다.

```
$ sudo gitlab-ci-multi-runner restart
```

클라우드 패키지 빌드

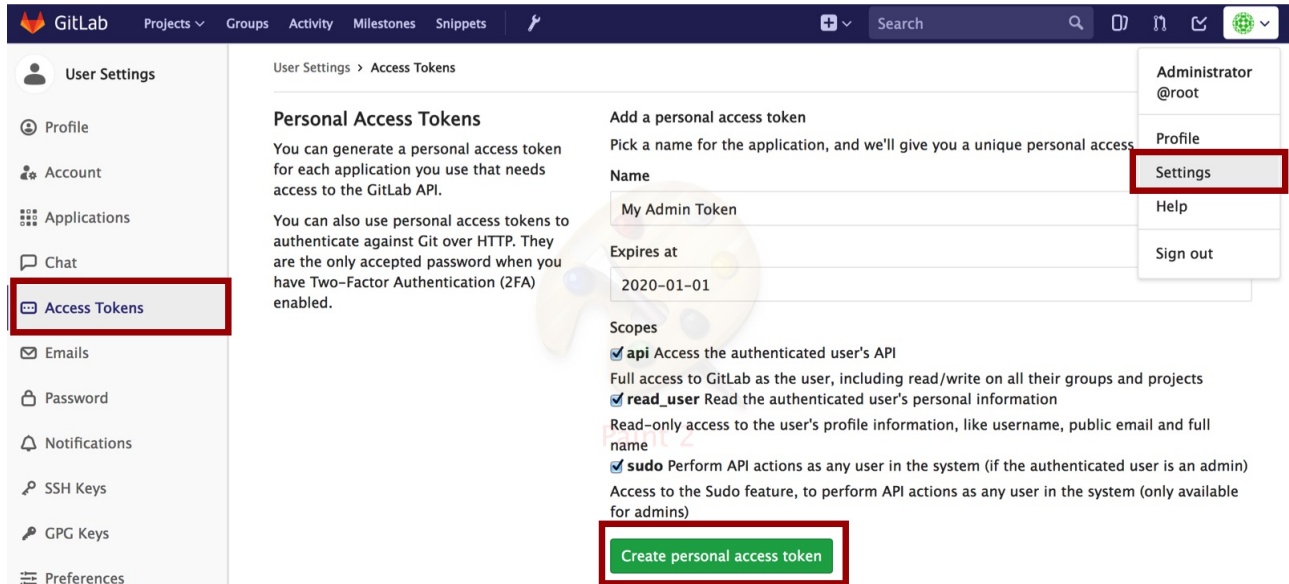
이제까지 진행해왔던 모든 설치 요소들이 정상적으로 설치되어있는지 확인하길 바랍니다.

- DC/OS
- Gitlab
- Gitlab CI
- Docker registry

지금부터는 Devops 운영 포털인 유엔진 클라우드 를 설치하게 되는데, DC/OS 클러스터와 깃랩 각각의 어드민 토큰이 필요합니다.

Get admin token

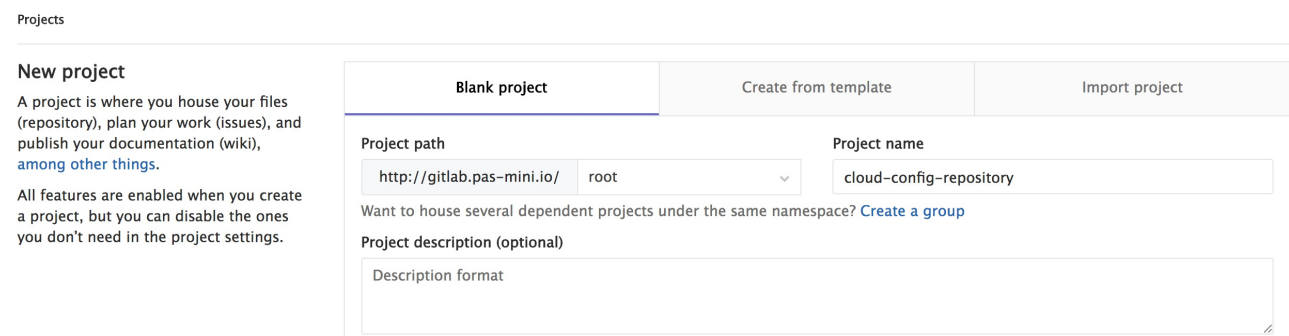
DC/OS 클러스터의 어드민 토큰 발급은 앞선 [사용자 생성 \(install-cluster-user.md\)](#) 에서 발급받은 토큰입니다. 깃랩 어드민 토큰은 깃랩 UI 로 접속하여 받을 수 있습니다. 먼저, 루트계정으로 깃랩으로 접속하도록 합니다. 초기 유저,패스워드는 root , adminadmin 입니다.



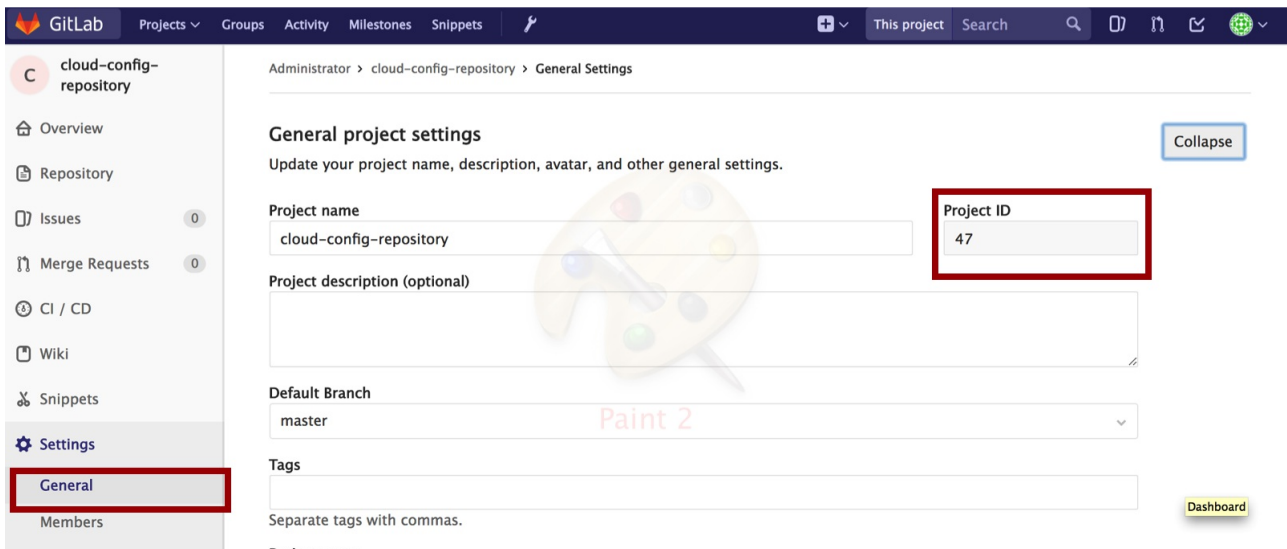
Create Gitlab Projects

클라우드 패키지 설치를 위해서 깃랩에 root 계정으로 다음의 프로젝트를 생성합니다.

- cloud-config-repository
- template-iam
- template-springboot
- template-vuejs
- template-zuul



cloud-config-repository 프로젝트는 데이터베이스의 역할을 수행하므로, [Fill out Cloud package part](#) 에서 이 프로젝트의 아이디가 필요합니다. UI 에서 cloud-config-repository 프로젝트 화면의 Settings => General project settings 메뉴로 들어가면 프로젝트 아이디를 볼 수 있습니다.



Fill out Cloud package part

클러스터 설치 ([install-cluster.md](#)) 에서 진행하였던 uEngine-cloud/uengine-resource/config.yml 파일의 Cloud package part 설정을 마무리 지어야 합니다.

본 파트에서 진행한 `dcos token`, `gitlab token`, `repository id` 를 사용하 다음 항목을들 채우도록 합니다.

```
# Cloud package part
# Fill out those properties after install DC/OS cluster && Gitlab
# Then, you should re-command "mvn clean install exec:java package"
dcos:
  token:
eyJhbGciOiJIUzI1NiIsInR5cCI6IkpXVCJ9.eyJ1aWQiOiJkYXJrZ29kYXJrZ29AZ21haWwY29tIiwiaXNjaXhwaXozMTI1NTU5MDY0MzkuODk0MX0.aHgH_M-g-n-WlnPg_CorMGYEprULPSeUTIGu3GyZQ-U

gitlab:
  root:
    username: root
    password: adminadmin
    token: -arWnfRY7S4h6oyRthNy
  config-repo:
    projectId: 47
    deployment-path: /deployment
    template-path: /template
```

메이븐 리소스를 통해 설정파일을 재생성합니다.

```
$ cd uEngine-cloud/uengine-resource
$ mvn clean install exec:java package
```

데이터 베이스 업로드

설정을 생성하고 나면, 다음과 같은 디렉토리 리스트를 볼 수 있습니다.

```

$ cd uEngine-cloud
$ ll

-rw-rw-r--. 1 centos centos 1074 1월 17 16:53 LICENSE
-rw-rw-r--. 1 centos centos 23620 1월 18 06:29 README.md
drwxrwxr-x. 5 centos centos 132 1월 17 17:01 cloud-config-repository
drwxrwxr-x. 2 centos centos 185 1월 17 17:01 deploys
drwxrwxr-x. 3 centos centos 4096 1월 18 06:29 document
drwxrwxr-x. 3 centos centos 4096 1월 17 18:59 install
-rwxrwxr-x. 1 centos centos 2463 1월 17 16:53 pom.xml
drwxrwxr-x. 5 centos centos 140 1월 17 16:53 template-iam
drwxrwxr-x. 4 centos centos 123 1월 17 16:53 template-springboot
drwxrwxr-x. 8 centos centos 4096 1월 17 16:53 template-vuejs
drwxrwxr-x. 4 centos centos 123 1월 17 16:53 template-zuul
drwxrwxr-x. 3 centos centos 50 1월 17 16:53 uengine-cloud-config
drwxrwxr-x. 5 centos centos 118 1월 17 16:53 uengine-cloud-iam
drwxrwxr-x. 3 centos centos 50 1월 17 16:53 uengine-cloud-server
drwxrwxr-x. 7 centos centos 259 1월 17 16:53 uengine-cloud-ui
drwxrwxr-x. 3 centos centos 50 1월 17 16:53 uengine-eureka-server
drwxrwxr-x. 8 centos centos 230 1월 18 06:31 uengine-resource

```

이 중 cloud-config-repository 과 template-* 로 시작되는 프로젝트들을 [Create Gitlab Projects](#) 단계에서 만들어 두었던 프로젝트로 업로드 해야 합니다.

업로드 중 유저이름, 패스워드를 묻는 프롬프트가 뜰 경우 깃랩 루트 유저네임과 패스워드를 입력하시면 됩니다.

```
$ cd cloud-config-repository
$ git init
$ git remote add origin http://gitlab.pas-mini.io/root/cloud-config-repository.git
$ git add .
$ git commit -m "Initial commit"
$ git push -u origin master

$ cd ../template-springboot
$ git init
$ git remote add origin http://gitlab.pas-mini.io/root/template-springboot.git
$ git add .
$ git commit -m "Initial commit"
$ git push -u origin master

$ cd ../template-iam
$ git init
$ git remote add origin http://gitlab.pas-mini.io/root/template-iam.git
$ git add .
$ git commit -m "Initial commit"
$ git push -u origin master

$ cd ../template-zuul
$ git init
$ git remote add origin http://gitlab.pas-mini.io/root/template-zuul.git
$ git add .
$ git commit -m "Initial commit"
$ git push -u origin master

$ cd ../template-vuejs
$ git init
$ git remote add origin http://gitlab.pas-mini.io/root/template-vuejs.git
$ git add .
$ git commit -m "Initial commit"
$ git push -u origin master
```

클라우드 패키지 빌드

uEngine-cloud/install 폴더에서, sh docker-build.sh 를 수행하도록 합니다.

이 스크립트는 config.yml 의 환경설정에 따라 커스터마이징 된 소스코드를 빌드하고, 도커 이미지 생성 후, 지정된 도커 레파지토리로 업로드합니다.

```
$ cd uEngine-cloud/install
$ sh docker-build.sh
```

클라우드 패키지 실행

uEngine-cloud/deloys 폴더에는 다음과 같은 파일들이 생성되 있을 것입니다.

```
$ cd uEngine-cloud/deloys
$ ll

-rw-rw-r--. 1 centos centos 1077 1월 18 06:31 cloud-server.json
-rw-rw-r--. 1 centos centos 1367 1월 18 06:31 cloud-ui.json
-rw-rw-r--. 1 centos centos 1656 1월 18 06:31 config.json
-rw-rw-r--. 1 centos centos 1665 1월 18 06:31 db.json
-rw-rw-r--. 1 centos centos 1457 1월 18 06:31 eureka-server.json
-rw-rw-r--. 1 centos centos 1061 1월 18 06:31 iam.json
-rw-rw-r--. 1 centos centos 34932 1월 18 06:31 marathon-lb-internal.json
-rw-rw-r--. 1 centos centos 34885 1월 18 06:31 marathon-lb.json
```

이 파일들은 DC/OS 클러스터에 배포하기 위한 [marathon-parameters](https://docs.mesosphere.com/1.10/deploying-services/marathon-parameters/#example) (<https://docs.mesosphere.com/1.10/deploying-services/marathon-parameters/#example>) 파일입니다.

다음 순차로 배포를 시작합니다.

```
# Run these three apps.
dcos marathon app add config.json
dcos marathon app add eureka-server.json
dcos marathon app add db.json
```

먼저, config.json , eureka-server.json , db.json 을 배포하도록 합니다.

- config.json : 클라우드 콘피그 서버
- eureka-server.json : 유레카 서버
- db.json : Mysql 데이터베이스

이후, DC/OS ui 로 접속하여 세가지 서비스가 모두 health 상태가 될때까지 기다리도록 합니다. 이 과정은 수분이 소요 될 수 있습니다.

혹시 Failed 된 Task 가 발생하게 된다면 로그를 확인 후 조치를 취하도록 합니다.

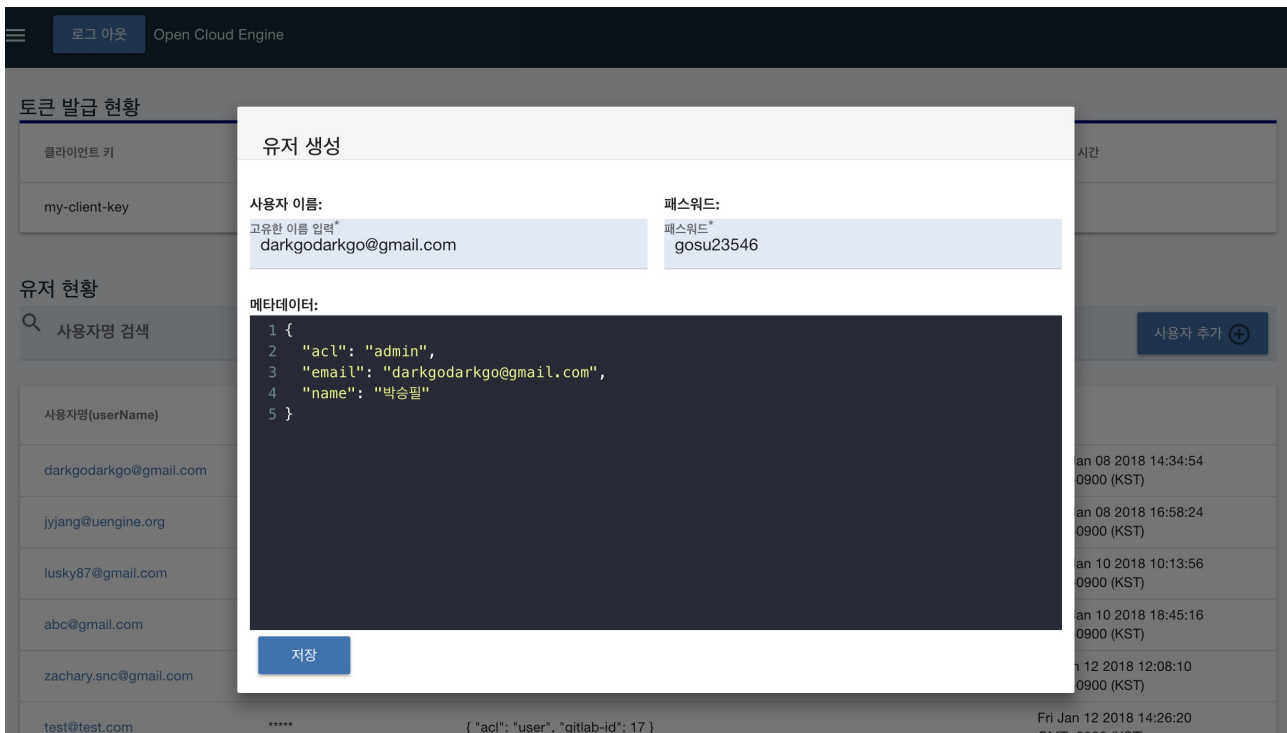
그 이후, 나머지 서비스들을 배포합니다.

```
# Check pre three apps are all healthy, then run under three apps.
dcos marathon app add iam.json
dcos marathon app add cloud-server.json
dcos marathon app add cloud-ui.json
```

나머지 서비스도 모두 health 상태가 되었다면, [도메인 && 네트워크 준비 \(pre-domain.md\)](#) 에서 설정한 iam 호스트 주소로 접속합니다. (ex. <http://iam.pas-mini.io>)

인증 서버의 로그인 창 아이디, 패스워드는 [클러스터 설치 - Security part \(install-cluster.md#security-part\)](#) 에서 설정한 정보입니다.

여기서 별다른 변경을 하지 않았다면, admin , admin 이 아이디, 패스워드 입니다. 로그인을 하시고 난 후 사용자 추가 버튼 클릭 후, 클라우드 플랫폼에 접속 가능한 인원을 편집하십시오.



여기서, metadata 항목의 acl 은 admin , user 가 있습니다. admin 은 모든 앱 및 클라우드 패키지, 그리고 도커 서비스를 편집할 수 있는 권한이 있고, user 는 자신이 생성한 앱, 또는 같은 깃랩 그룹에 소속된 인원의 앱을 편집할 수 있습니다.

테스트

Cadvisor Proxy Server

[클러스터 설치 \(install-cluster.md\)](#) 과정에서 Google 의 cadvisor 도커가 함께 설치되게 되는데, 이는 도커 컨테이너 메트릭스를 수집하여 APM , 모니터링 등에 이용할 수 있게 해주는 툴 입니다.

cadvisor 는 각 에이전트 노드마다 설치되는데, 유엔진 클라우드 플랫폼은 에이전트 서버로의 직접적인 접근을 허용하지 않기 때문에 public-agent 노드로 cadvisor 정보를 proxy 통신을 시켜주어야 합니다.

public-agent 노드로 이동한 후, 다음 설치과정을 진행합니다.

```
$ ssh -i <your-private-key> public
```

rinted 설치

```
$ sudo yum -y install gcc
$ wget http://www.boutell.com/rinetd/http/rinetd.tar.gz
$ tar zxvf rinetd.tar.gz
$ cd rinetd
$ sudo make
$ sudo mv rinetd /usr/sbin/
$ sudo chcon -u system_u -t bin_t /usr/sbin/rinetd
```

rinted 프록시 설정

cadvisor 서비스는 각 agent-node 의 8080 포트를 통해 서비스 됩니다. 모든 에이전트 노드의 8080 서버들이 public-agent 의 90** 포트로 프록시

통신이 되도록 설정합니다. 예를 들어, agent11 번인 경우, 9011 포트로 설정합니다.

```
$ sudo vi /etc/rinetd.conf

logfile /var/log/rinetd/rinetd.log
logcommon

#src IP, Port, dest IP, Port
0.0.0.0 9100 172.31.5.136 8080 # public agent is 9100
0.0.0.0 9101 172.31.6.35 8080 # other agent is 91** (agent number)
0.0.0.0 9102 172.31.1.235 8080
0.0.0.0 9103 172.31.5.245 8080
0.0.0.0 9104 172.31.14.247 8080
0.0.0.0 9105 172.31.7.160 8080
0.0.0.0 9106 172.31.11.70 8080
0.0.0.0 9107 172.31.0.164 8080
```

rinted 서비스 등록

/etc/rc.d/init.d/rinetd 생성

```
$ sudo vi /etc/rc.d/init.d/rinetd

#!/bin/sh
#
# chkconfig: - 80 20
# description: rinetd is a TCP redirection server
# processname: rinetd
# pidfile: /var/run/rinetd.pid
# config: /etc/rinetd.conf

# Source function library.
. /etc/rc.d/init.d/functions

# Source networking configuration.
. /etc/sysconfig/network

# Check that networking is up.
[ ${NETWORKING} = "no" ] && exit 1

exec="/usr/sbin/rinetd"
prog=$(basename $exec)

lockfile=/var/lock/subsys/$prog

start() {
    echo -n $"Starting $prog: "
```

```
daemon $exec
    retval=$?
    echo
    [ $retval -eq 0 ] && touch $lockfile
    return $retval
}

stop() {
    echo -n "$Stopping $prog: "
    killproc $prog
    retval=$?
    echo
    [ $retval -eq 0 ] && rm -f $lockfile
    return $retval
}

restart() {
    stop
    start
}

reload() {
    if [ -f "$lockfile" ]; then
        echo -n "$Reloading $prog: "
        killproc $prog -HUP
        retval=$?
        echo
        return $retval
    else
        restart
    fi
}

force_reload() {
    restart
}

fdr_status() {
    status $prog
}

case "$1" in
    start|stop|restart|reload)
        $1
        ;;
    force-reload)
        force_reload
        ;;
    status)
        fdr_status
        ;;
    condrestart|try-restart)
```

```

    [ ! -f $lockfile ] || restart
;;
*)
    echo $"Usage: $0 {start|stop|status|restart|try-restart|reload|force-
reload}"
    exit 2
esac

```

서비스 등록

```

$ sudo chcon -u system_u -t initrc_exec_t /etc/rc.d/init.d/rinetd
$ sudo chmod +x /etc/rc.d/init.d/rinetd
$ sudo chkconfig rinetd on

```

/etc/logrotate.d/rinetd 로그 파일 생성

```

$ sudo vi /etc/logrotate.d/rinetd

/var/log/rinetd.log {
    missingok
    notifempty
    delaycompress
    postrotate
        /sbin/service rinetd restart 2> /dev/null > /dev/null || true
    endscript
}

```

rinetd 서비스 최종 실행

```

$ sudo mkdir /var/log/rinetd
$ sudo chcon -u system_u /var/log/rinetd
$ sudo service rinetd start

```

이후, 깃랩 UI 의 cloud-config-repository 프로젝트의 uengine-cloud-server.yml 파일에 등록한 노드만큼 <추가아이피>, http://<퍼블릭노드아이피>:91** 형식으로 더해줍니다.

```

cadvisor:
- 172.31.5.136,http://52.79.51.79:9100
- 172.31.6.35,http://52.79.51.79:9101
- 172.31.1.235,http://52.79.51.79:9102
- 172.31.5.245,http://52.79.51.79:9103
- 172.31.14.247,http://52.79.51.79:9104
- 172.31.7.160,http://52.79.51.79:9105
- 172.31.11.70,http://52.79.51.79:9106
- 172.31.0.164,http://52.79.51.79:9107

```

모든 과정이 종료된 후, 클라우드 플랫폼에서 각 앱의 타스크(컨테이너) 상세보기를 통해 cadvisor 메트릭스를 볼 수 있습니다.

노드 추가

이 파트는 **private-agent** 를 추가하는 작업을 기술합니다. **master** 노드를 추가 / 삭제하는 것은 기존 서비스를 중단하지 않고 수행하는 것은 매우 어려운 절차를 가지므로 플랫폼 배포사에 문의를 하시기 바랍니다.

uEngine-cloud/uengine-resource/config.yml 파일에 add-agent 항목에, 추가할 노드 리스트를 기술합니다.

```
$ vi uEngine-cloud/config.yml

.
.
  add-agent:
    agent8: 172.31.10.202
    agent9: 172.31.14.197
    agent10: 172.31.13.131
    agent11: 172.31.1.166
    agent12: 172.31.8.5
```

uEngine-cloud/uengine-resource/config.yml 을 작성 후, 빌드를 실행합니다. 이 단계를 실행하면 DC/OS 컨피그레이션 및 앤서블 스크립트가 재생성됩니다.

```
$ cd uengine-resource
$ mvn clean install exec:java package
```

이후, uEngine-cloud/install 폴더에서 변경 적용된 DC/OS 클러스터 배포파일을 생성 후, 압축합니다.

```
# 앤서블 호스트 파일 재적용
$ cd install
$ sudo sh -c "cat ansible-hosts.yml > /etc/ansible/hosts"

# DC/OS 생성파일 생성
$ sudo bash dcos_generate_config.sh --genconf

# DC/OS 배포파일 압축
$ cd genconf/serve
sudo tar cf dcos-install.tar *
```

다시 uEngine-cloud/install 폴더로 돌아와, 마지막으로 ansible-add-agent.yml 플레이북을 실행합니다.

```
$ cd install
$ ansible-playbook ansible-add-agent.yml
```

cadvisor 프록시 추가

Google cadvisor 서비스를 사용하기 위해 public node 의 rinted 프록시 서버를 설정합니다.

추가하는 순서는 agent 번호 순번대로, 91** 으로 만들어주면 됩니다. 예를들어 agent12 번일 경우, 9112 포트가 됩니다.

```
$ ssh -i <your-private-key> public
```

```
$ sudo vi /etc/rinetd.conf
```

```
.  
.
0.0.0.0 9108 172.31.10.202 8080
0.0.0.0 9109 172.31.14.197 8080
0.0.0.0 9110 172.31.13.131 8080
0.0.0.0 9111 172.31.1.166 8080
0.0.0.0 9112 172.31.8.5 8080
```

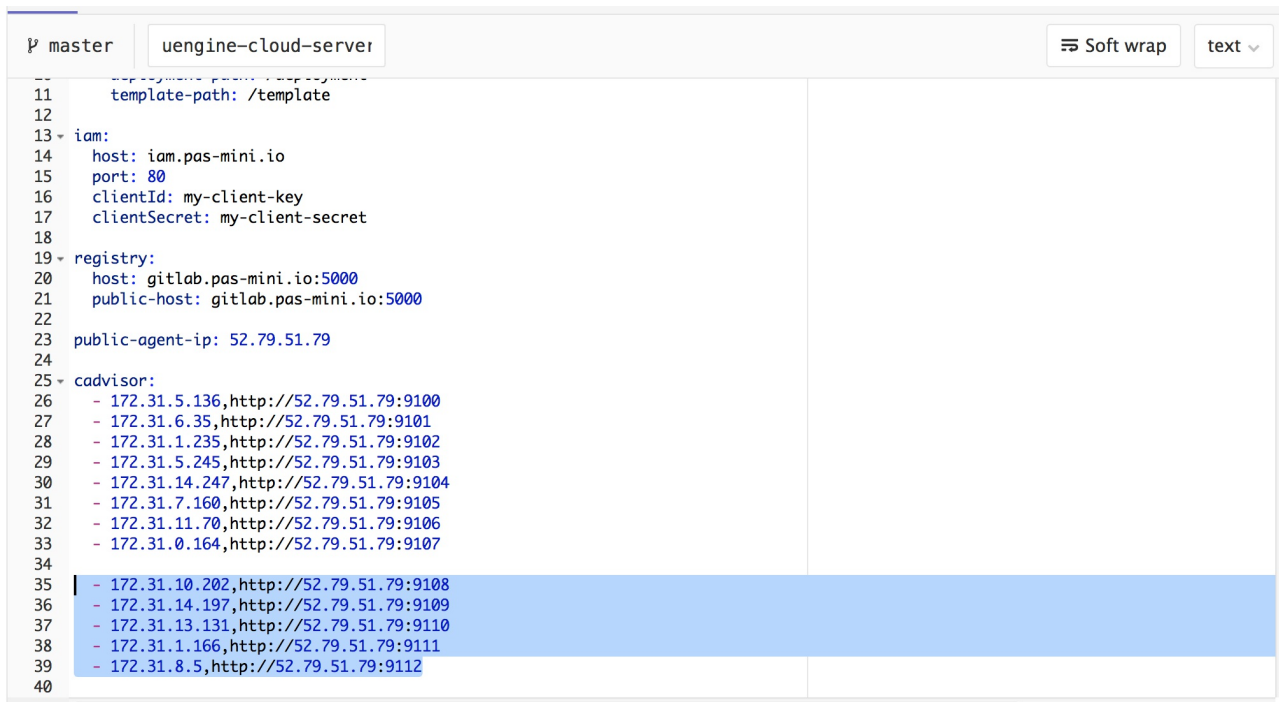
rinted 프록시 서버를 재시작합니다.

```
$ sudo service rinetd restart
```

```
Restarting rinetd (via systemctl):
```

```
[ OK ]
```

이후, 깃랩 UI 의 cloud-config-repository 프로젝트의 uengine-cloud-server.yml
파일에 추가된 노드만큼 <추가아이피>,http://<퍼블릭노드아이피>:91** 형식으로 더해줍니다.



```
11 template-path: /template
12
13 iam:
14   host: iam.pas-mini.io
15   port: 80
16   clientId: my-client-key
17   clientSecret: my-client-secret
18
19 registry:
20   host: gitlab.pas-mini.io:5000
21   public-host: gitlab.pas-mini.io:5000
22
23 public-agent-ip: 52.79.51.79
24
25 cadvisor:
26   - 172.31.5.136,http://52.79.51.79:9100
27   - 172.31.6.35,http://52.79.51.79:9101
28   - 172.31.1.235,http://52.79.51.79:9102
29   - 172.31.5.245,http://52.79.51.79:9103
30   - 172.31.14.247,http://52.79.51.79:9104
31   - 172.31.7.160,http://52.79.51.79:9105
32   - 172.31.11.70,http://52.79.51.79:9106
33   - 172.31.0.164,http://52.79.51.79:9107
34
35   - 172.31.10.202,http://52.79.51.79:9108
36   - 172.31.14.197,http://52.79.51.79:9109
37   - 172.31.13.131,http://52.79.51.79:9110
38   - 172.31.1.166,http://52.79.51.79:9111
39   - 172.31.8.5,http://52.79.51.79:9112
40
```

cadvisor:

```
.
.
.
- 172.31.10.202,http://52.79.51.79:9108
- 172.31.14.197,http://52.79.51.79:9109
- 172.31.13.131,http://52.79.51.79:9110
- 172.31.1.166,http://52.79.51.79:9111
- 172.31.8.5,http://52.79.51.79:9112
```

노드 삭제

노드를 제거하기 위해서는, 운영 중인 앱이 안전하게 다른 노드로 이전되고 난 후 제거되어야 합니다. `gracefully-remove-agent` 가 진행되기 위해서, `uEngine-cloud/uengine-resource/config.yml` 파일에 `gracefully-remove-agent` 항목에, 제거할 노드 리스트를 기술합니다.

```
$ vi uEngine-cloud/config.yml

.
.
  gracefully-remove-agent:
    agent9: 172.31.14.197
    agent10: 172.31.13.131
    agent11: 172.31.1.166
    agent12: 172.31.8.5
```

`uEngine-cloud/uengine-resource/config.yml` 을 작성 후, 빌드를 실행합니다. 이 단계를 실행하면 DC/OS 컨피그레이션 및 앤서블 스크립트가 재생성됩니다.

```
$ cd uengine-resource
$ mvn clean install exec:java package
```

이후, `uEngine-cloud/install` 폴더에서 변경 적용된 DC/OS 클러스터 배포파일을 생성합니다. ([노드 추가 \(op-add-node.md\)](#)와 다르게, 압축 파일을 생성하지 않아도 됩니다.)

```
# 앤서블 호스트 파일 재적용
$ cd install
$ sudo sh -c "cat ansible-hosts.yml > /etc/ansible/hosts"
```

다시 `uEngine-cloud/install` 폴더로 돌아와, 마지막으로 `ansible-gracefully-remove-agent.yml` 플 레이북을 실행합니다.

```
$ cd install
$ ansible-playbook ansible-gracefully-remove-agent.yml
```

클라우드 플랫폼에서, 기존 TASK들이 남아있는 노드로 이동된 후, 노드가 삭제되는 모습을 볼 수 있습니다.

설치 완전 삭제

설치를 완전히 제거하는 것은 다음과 같은 경우에 해당합니다.

- 일부 `private-agent` 또는 `public-agent` 에 대하여 설치요소를 완전히 삭제할 경우
- [클러스터 인스톨 \(install-cluster.md\)](#) 과정에서 잘못된 설정값으로 초기 설치를 진행하여, 수정 후에도 실패를 경험하는 경우
- 클러스터의 완전 삭제가 필요한 경우

노드 완전 삭제가 진행되기 위해서, `uEngine-cloud/uengine-resource/config.yml` 파일의 `uninstall` 항목에, 제거할 노드 리스트를 기술합니다.

```
$ vi uEngine-cloud/uengine-resource/config.yml
```

```
.  
.
  
  uninstall:  
    public: 172.31.5.136  
    master1: 172.31.12.143  
    master2: 172.31.4.125  
    master3: 172.31.1.198  
    agent1: 172.31.6.35  
    agent2: 172.31.1.235  
    agent3: 172.31.5.245  
    agent4: 172.31.14.247  
    agent5: 172.31.7.160  
    agent6: 172.31.11.70  
    agent7: 172.31.0.164  
    agent9: 172.31.14.197  
    agent10: 172.31.13.131  
    agent11: 172.31.1.166  
    agent12: 172.31.8.5
```

uEngine-cloud/uengine-resource/config.yml 을 작성 후, 빌드를 실행합니다. 이 단계를 실행하면 DC/OS 컨피그레이션 및 앤서블 스크립트가 재생성됩니다.

```
$ cd uengine-resource  
$ mvn clean install exec:java package
```

이후, uEngine-cloud/install 폴더에서 변경 적용된 DC/OS 클러스터 배포파일을 생성합니다.
([노드 추가 \(op-add-node.md\)](#)와 다르게, 압축 파일을 생성하지 않아도 됩니다.)

```
# 앤서블 호스트 파일 재적용  
$ cd install  
$ sudo sh -c "cat ansible-hosts.yml > /etc/ansible/hosts"
```

다시 uEngine-cloud/install 폴더로 돌아와, 마지막으로 ansible-uninstall.yml 플레이북을 실행합니다.

```
$ cd install  
$ ansible-playbook ansible-uninstall.yml
```

깃랩 백업

백업 설정

깃랩 데이터 백업은 데이터베이스가 포함 된 아카이브 파일을 생성하고, 모든 레파지토리 및 모든 첨부 파일, Artifacts 를 포함합니다.

백업은 정확히 동일한 버전 및 유형 (CE / EE)으로 만 복원 할 수 있습니다. 동일한 버전끼리의 깃랩 서버간에는 백업, 리 스토어 수행이 자유롭습니다.

rsync 설치

다음 명령어로 rsync 를 설치합니다.


```
$ sudo yum install rsync -y
```

백업 타임스탬프

백업 아카이브 파일은 config/gitlab.yml 에 지정된 backup_path 에 저장됩니다.

파일 이름은 [TIMESTAMP]_gitlab_backup.tar 이며, 여기서 TIMESTAMP 로 백업이 생성 된 시간과 GitLab 버전을 식별합니다.

예를 들어 백업 이름이 1493107454_2017_04_25_9.1.0_gitlab_backup.tar 인 경우, 타임스탬프는 1493107454_2017_04_25_9.1.0 입니다.

백업 만들기

백업을 수행하기 위해 깃랩 서버를 내릴 필요는 없습니다. 다음 명령어로 백업을 생성합니다.

```
$ sudo gitlab-rake gitlab:backup:create

Dumping database tables:
- Dumping table events... [DONE]
- Dumping table issues... [DONE]
- Dumping table keys... [DONE]
- Dumping table merge_requests... [DONE]
- Dumping table milestones... [DONE]
- Dumping table namespaces... [DONE]
- Dumping table notes... [DONE]
- Dumping table projects... [DONE]
- Dumping table protected_branches... [DONE]
- Dumping table schema_migrations... [DONE]
- Dumping table services... [DONE]
- Dumping table snippets... [DONE]
- Dumping table taggings... [DONE]
- Dumping table tags... [DONE]
- Dumping table users... [DONE]
- Dumping table users_projects... [DONE]
- Dumping table web_hooks... [DONE]
- Dumping table wikis... [DONE]
Dumping repositories:
- Dumping repository abcd... [DONE]
Creating backup archive: $TIMESTAMP_gitlab_backup.tar [DONE]
Deleting tmp directories...[DONE]
Deleting old backups... [SKIPPING]
```

Uploading backups to a remote (cloud) storage

GitLab 7.4부터 백업 스크립트가 생성 한 .tar 파일을 업로드하도록 할 수 있도록 업데이트 되었습니다.

아래 예제에서는 Amazon S3 에 저장되도록 설정하지만, 다음과 같은 백업 저장소를 선택할 수 있습니다.

- [imports cloud drivers \(https://gitlab.com/gitlab-org/gitlab-ce/blob/30f5b9a5b711b46f1065baf755e413ceced5646b/Gemfile#L88\)](https://gitlab.com/gitlab-org/gitlab-ce/blob/30f5b9a5b711b46f1065baf755e413ceced5646b/Gemfile#L88) for AWS, Google, OpenStack Swift, Rackspace and Aliyun as well.

- A local driver is [also available](#).

Using Amazon S3

옴니버스 패키지 설정:

1. /etc/gitlab/gitlab.rb 에 다음과 같이 추가합니다. :

```
gitlab_rails['backup_upload_connection'] = {  
  'provider' => 'AWS',  
  'region' => 'eu-west-1',  
  'aws_access_key_id' => 'AKIAKIAKI',  
  'aws_secret_access_key' => 'secret123'  
  # If using an IAM Profile, don't configure aws_access_key_id &  
  aws_secret_access_key  
  # 'use_iam_profile' => true  
}  
gitlab_rails['backup_upload_remote_directory'] = 'my.s3.bucket'
```

2. 바뀐 설정을 적용합니다.

```
$ sudo gitlab-ctl reconfigure
```

uploading-to-locally-mounted-shares

백업저장소로 (NFS / CIFS / SMB / etc.) 등과 같은 마운트 디스크를 통해 수행할 수 있는데, 이때는 Fog [Local \(https://github.com/fog/fog-local#usage\)](https://github.com/fog/fog-local#usage) 스토리지 프로바이더를 사용하게 됩니다.

The directory pointed to by the `local_root` key **must** be owned by the git user **when mounted** (mounting with the `uid=` of the git user for CIFS and SMB) or the user that you are executing the backup tasks under (for omnibus packages, this is the git user).

The `backup_upload_remote_directory` **must** be set in addition to the `local_root` key. This is the sub directory inside the mounted directory that backups will be copied to, and will be created if it does not exist. If the directory that you want to copy the tarballs to is the root of your mounted directory, just use `.` instead.

옴니버스 패키지 설정: (설정 후 `$ sudo gitlab-ctl reconfigure` 를 수행하세요.)

```
gitlab_rails['backup_upload_connection'] = {  
  :provider => 'Local',  
  :local_root => '/mnt/backups'  
}  
  
# The directory inside the mounted folder to copy backups to  
# Use '.' to store them in the root directory  
gitlab_rails['backup_upload_remote_directory'] = 'gitlab_backups'
```

Specifying a custom directory for backups

If you want to group your backups you can pass a `DIRECTORY` environment variable:

```
sudo gitlab-rake gitlab:backup:create DIRECTORY=daily
sudo gitlab-rake gitlab:backup:create DIRECTORY=weekly
```

Backup archive permissions

The backup archives created by GitLab (1393513186_2014_02_27_gitlab_backup.tar) will have owner/group git:git and 0600 permissions by default.

This is meant to avoid other system users reading GitLab's data.

If you need the backup archives to have different permissions you can use the 'archive_permissions' setting.

```
# In /etc/gitlab/gitlab.rb, for omnibus packages
gitlab_rails['backup_archive_permissions'] = 0644 # Makes the backup archives
world-readable
```

Storing configuration files

Please be informed that a backup does not store your configuration files. One reason for this is that your database contains encrypted information for two-factor authentication. Storing encrypted information along with its key in the same place defeats the purpose of using encryption in the first place!

If you use an Omnibus package please see the [instructions in the readme to backup your configuration](https://gitlab.com/gitlab-org/omnibus-gitlab/blob/master/README.md#backup-and-restore-omnibus-gitlab-configuration) (<https://gitlab.com/gitlab-org/omnibus-gitlab/blob/master/README.md#backup-and-restore-omnibus-gitlab-configuration>).

If you have a cookbook installation there should be a copy of your configuration in Chef. If you installed from source, please consider backing up your config/secrets.yml file, gitlab.yml file, any SSL keys and certificates, and your [SSH host keys](https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079) (<https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079>).

At the very **minimum** you should backup /etc/gitlab/gitlab.rb and /etc/gitlab/gitlab-secrets.json (Omnibus), or /home/git/gitlab/config/secrets.yml (source) to preserve your database encryption key.

Configuring cron to make daily backups

Note:

The following cron jobs do not [backup your GitLab configuration files](https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079) or [SSH host keys](https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079) (<https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079>).

For Omnibus installations

To schedule a cron job that backs up your repositories and GitLab metadata, use the root user:

```
sudo su -
crontab -e
```

There, add the following line to schedule the backup for everyday at 2 AM:

```
0 2 * * * /opt/gitlab/bin/gitlab-rake gitlab:backup:create CRON=1
```

You may also want to set a limited lifetime for backups to prevent regular backups using all your disk space. To do this add the following lines to `/etc/gitlab/gitlab.rb` and reconfigure:

```
# limit backup lifetime to 7 days - 604800 seconds
gitlab_rails['backup_keep_time'] = 604800
```

Note that the `backup_keep_time` configuration option only manages local files. GitLab does not automatically prune old files stored in a third-party object storage (e.g., AWS S3) because the user may not have permission to list and delete files. We recommend that you configure the appropriate retention policy for your object storage. For example, you can configure [the S3 backup policy as described here \(http://stackoverflow.com/questions/37553070/gitlab-omnibus-delete-backup-from-amazon-s3\)](http://stackoverflow.com/questions/37553070/gitlab-omnibus-delete-backup-from-amazon-s3).

백업 리스토어

GitLab provides a simple command line interface to restore your whole installation, and is flexible enough to fit your needs.

The [restore prerequisites section](#) includes crucial information. Make sure to read and test the whole restore process at least once before attempting to perform it in a production environment.

You can only restore a backup to **exactly the same version and type (CE/EE)** of GitLab that you created it on, for example CE 9.1.0.

Restore prerequisites

You need to have a working GitLab installation before you can perform a restore. This is mainly because the system user performing the restore actions ('git') is usually not allowed to create or delete the SQL database it needs to import data into ('gitlabhq_production'). All existing data will be either erased (SQL) or moved to a separate directory (repositories, uploads).

To restore a backup, you will also need to restore `/etc/gitlab/gitlab-secrets.json` (for Omnibus packages) or `/home/git/gitlab/.secret` (for installations from source).

If you fail to restore this encryption key file along with the application data backup, users with two-factor authentication enabled and GitLab Runners will lose access to your GitLab server.

Depending on your case, you might want to run the restore command with one or more of the following options:

- `BACKUP=timestamp_of_backup` – Required if more than one backup exists. Read what the [backup timestamp is about](#).
- `force=yes` – Do not ask if the `authorized_keys` file should get regenerated.

Restore for Omnibus installations

This procedure assumes that:

- You have installed the **exact same version and type (CE/EE)** of GitLab Omnibus with which the backup was created.
- You have run `sudo gitlab-ctl reconfigure` at least once.
- GitLab is running. If not, start it using `sudo gitlab-ctl start`.

First make sure your backup tar file is in the backup directory described in the `gitlab.rb` configuration `gitlab_rails['backup_path']`. The default is `/var/opt/gitlab/backups/`.

```
sudo cp 1493107454_2017_04_25_9.1.0_gitlab_backup.tar /var/opt/gitlab/backups/
```

Stop the processes that are connected to the database. Leave the rest of GitLab running:

```
sudo gitlab-ctl stop unicorn
sudo gitlab-ctl stop sidekiq
# Verify
sudo gitlab-ctl status
```

Next, restore the backup, specifying the timestamp of the backup you wish to restore:

```
# This command will overwrite the contents of your GitLab database!
sudo gitlab-rake gitlab:backup:restore BACKUP=1493107454_2017_04_25_9.1.0
```

Next, restore `/etc/gitlab/gitlab-secrets.json` if necessary as mentioned above.

Restart and check GitLab:

```
sudo gitlab-ctl restart
sudo gitlab-rake gitlab:check SANITIZE=true
```

If there is a GitLab version mismatch between your backup tar file and the installed version of GitLab, the restore command will abort with an error. Install the [correct GitLab version \(https://packages.gitlab.com/gitlab/\)](https://packages.gitlab.com/gitlab/) and try again.

Alternative backup strategies

If your GitLab server contains a lot of Git repository data you may find the GitLab backup script to be too slow.

In this case you can consider using filesystem snapshots as part of your backup strategy.

Example: Amazon EBS

```
A GitLab server using omnibus-gitlab hosted on Amazon AWS.
An EBS drive containing an ext4 filesystem is mounted at /var/opt/gitlab.
In this case you could make an application backup by taking an EBS snapshot.
The backup includes all repositories, uploads and Postgres data.
```

Example: LVM snapshots + rsync

A GitLab server using omnibus-gitlab, with an LVM logical volume mounted at /var/opt/gitlab.
Replicating the /var/opt/gitlab directory using rsync would not be reliable because too many files would change while rsync is running.
Instead of rsync-ing /var/opt/gitlab, we create a temporary LVM snapshot, which we mount as a read-only filesystem at /mnt/gitlab_backup.
Now we can have a longer running rsync job which will create a consistent replica on the remote server.
The replica includes all repositories, uploads and Postgres data.

If you are running GitLab on a virtualized server you can possibly also create VM snapshots of the entire GitLab server.

It is not uncommon however for a VM snapshot to require you to power down the server, so this approach is probably of limited practical use.

Troubleshooting

Restoring database backup using omnibus packages outputs warnings

If you are using backup restore procedures you might encounter the following warnings:

```
psql:/var/opt/gitlab/backups/db/database.sql:22: ERROR: must be owner of
extension plpgsql
psql:/var/opt/gitlab/backups/db/database.sql:2931: WARNING: no privileges
could be revoked for "public" (two occurrences)
psql:/var/opt/gitlab/backups/db/database.sql:2933: WARNING: no privileges were
granted for "public" (two occurrences)
```

Be advised that, backup is successfully restored in spite of these warnings.

The rake task runs this as the gitlab user which does not have the superuser access to the database. When restore is initiated it will also run as gitlab user but it will also try to alter the objects it does not have access to.

Those objects have no influence on the database backup/restore but they give this annoying warning.

For more information see similar questions on postgresql issue tracker [here](http://www.postgresql.org/message-id/201110220712.30886.adrian.klaver@gmail.com) (<http://www.postgresql.org/message-id/201110220712.30886.adrian.klaver@gmail.com>) and [here](http://www.postgresql.org/message-id/2039.1177339749@sss.pgh.pa.us) (<http://www.postgresql.org/message-id/2039.1177339749@sss.pgh.pa.us>) as well as [stack overflow](http://stackoverflow.com/questions/4368789/error-must-be-owner-of-language-plpgsql) (<http://stackoverflow.com/questions/4368789/error-must-be-owner-of-language-plpgsql>).

깃랩 환경설정 백업

Backup and restore Omnibus GitLab configuration

It is recommended to keep a copy of /etc/gitlab, or at least of /etc/gitlab/gitlab-secrets.json, in a safe place. If you ever need to restore a GitLab application backup you need to also restore

gitlab-secrets.json. If you do not, GitLab users who are using two-factor authentication will lose access to your GitLab server and 'secure variables' stored in GitLab CI will be lost.

It is not recommended to store your configuration backup in the same place as your application data backup, see below.

All configuration for omnibus-gitlab is stored in /etc/gitlab. To backup your configuration, just backup this directory.

```
# Example backup command for /etc/gitlab:
# Create a time-stamped .tar file in the current directory.
# The .tar file will be readable only to root.
sudo sh -c 'umask 0077; tar -cf $(date "+etc-gitlab-%s.tar") -C / etc/gitlab'
```

To create a daily application backup, edit the cron table for user root:

```
sudo crontab -e -u root
```

The cron table will appear in an editor.

Enter the command to create a compressed tar file containing the contents of /etc/gitlab/. For example, schedule the backup to run every morning after a weekday, Tuesday (day 2) through Saturday (day 6):

```
15 04 * * 2-6 umask 0077; tar cfz /secret/gitlab/backups/$(date "+etc-gitlab-
\s.tgz") -C / etc/gitlab
```

[cron is rather particular \(http://www.pantz.org/software/cron/croninfo.html\)](http://www.pantz.org/software/cron/croninfo.html)

about the cron table. Note:

- The empty line after the command
- The escaped percent character: %

You can extract the .tar file as follows.

```
# Rename the existing /etc/gitlab, if any
sudo mv /etc/gitlab /etc/gitlab.$(date +%s)
# Change the example timestamp below for your configuration backup
sudo tar -xf etc-gitlab-1399948539.tar -C /
```

Remember to run `sudo gitlab-ctl reconfigure` after restoring a configuration backup.

NOTE: Your machines SSH host keys are stored in a separate location at /etc/ssh/. Be sure to also [backup and restore those keys \(https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079\)](https://superuser.com/questions/532040/copy-ssh-keys-from-one-server-to-another-server/532079#532079) to avoid man-in-the-middle attack warnings if you have to perform a full machine restore.

Separate configuration backups from application data

Do not store your GitLab application backups (Git repositories, SQL data) in the same place as your configuration backup (/etc/gitlab). The gitlab-secrets.json file (and possibly also the gitlab.rb

file) contain database encryption keys to protect sensitive data in the SQL database:

- GitLab two-factor authentication (2FA) user secrets ('QR codes')
- GitLab CI 'secure variables'

If you separate your configuration backup from your application data backup, you reduce the chance that your encrypted application data will be lost/leaked/stolen together with the keys needed to decrypt it.

Creating an application backup

To create a backup of your repositories and GitLab metadata, follow the [backup create documentation](https://docs.gitlab.com/ce/raketasks/backup_restore.html#creating-a-backup-of-the-gitlab-system) (https://docs.gitlab.com/ce/raketasks/backup_restore.html#creating-a-backup-of-the-gitlab-system).

Backup create will store a tar file in `/var/opt/gitlab/backups`.

If you want to store your GitLab backups in a different directory, add the following setting to `/etc/gitlab/gitlab.rb` and run `sudo gitlab-ctl reconfigure`:

```
gitlab_rails['backup_path'] = '/mnt/backups'
```

Restoring an application backup

See [backup restore documentation](https://docs.gitlab.com/ce/raketasks/backup_restore.html#restore-for-omnibus-installations) (https://docs.gitlab.com/ce/raketasks/backup_restore.html#restore-for-omnibus-installations).

Upload backups to remote (cloud) storage

For details check [backup restore document of GitLab CE](https://docs.gitlab.com/ce/raketasks/backup_restore.html#uploading-backups-to-a-remote-cloud-storage) (https://docs.gitlab.com/ce/raketasks/backup_restore.html#uploading-backups-to-a-remote-cloud-storage).

Manually manage backup directory

Omnibus-gitlab creates the backup directory set with `gitlab_rails['backup_path']`. The directory is owned by the user that is running GitLab and it has strict permissions set to be accessible to only that user.

That directory will hold backup archives and they contain sensitive information.

In some organizations permissions need to be different because of, for example, shipping the backup archives offsite.

To disable backup directory management, in `/etc/gitlab/gitlab.rb` set:

```
gitlab_rails['manage_backup_path'] = false
```

Warning If you set this configuration option, it is up to you to create the directory specified in `gitlab_rails['backup_path']` and to set permissions which will allow user specified in `user['username']` to have correct access. Failing to do so will prevent GitLab from creating the backup archive.

트러블 슈팅

- [컴포넌트 헬스 체크 \(trouble-component.md\)](#)
- [네트워크 \(trouble-network.md\)](#)

트러블 슈팅 - 컴포넌트 구동 확인

- DC/OS [트러블 슈팅 \(https://dcos.io/docs/1.10/installing/troubleshooting/\)](https://dcos.io/docs/1.10/installing/troubleshooting/) 문서

아래는 DC/OS 를 구성하는 컴포넌트들의 로그를 볼 수 있는 명령어로, 컴포넌트의 동작상태를 tailing 합니다.

```
journalctl -flu dcos-exhibitor
journalctl -flu dcos-spartan
journalctl -flu dcos-mesos-dns
journalctl -flu dcos-mesos-slave
journalctl -flu dcos-mesos-slave-public
journalctl -flu dcos-mesos-master
journalctl -flu dcos-gen-resolvconf
```

트러블 슈팅 - 네트워크

유엔진 클라우드 플랫폼의 앱들의 최종 라우팅은 Haproxy 로써, DC/OS 에서 제공하는 marathon-lb 오픈소스입니다.

앱 생성 후 라우터 접속이 원활하지 않은 경우, Haproxy 의 설정 및 네트워크 상황을 다음의 레퍼런스를 통하여 체크할 수 있습니다.

Statistics	<public-node>:9090/haproxy?stats
Statistics CSV	<public-node>:9090/haproxy?stats;csv
Health check	<public-node>:9090/_haproxy_health_check
Configuration file view	<public-node>:9090/_haproxy_getconfig
Get vHost to backend map	<public-node>:9090/_haproxy_getvhostmap
Get app ID to backend map	<public-node>:9090/_haproxy_getappmap
Reload configuration	<public-node>:9090/_mlb_signal/hup*