



Porting Manual

개발 환경

클러스터 사양

EC2 서버 인스턴스 사양

UI / UX

형상 관리

OS

이슈 관리

Communication

Front-end

DB

Back-end

IDE

Infra

기타 편의 툴

Monitoring

배포 과정

1. SSAFY EC2(서버) 접속

1. WSL 사용하여 EC2에 SSH 연결

2. EC2 초기 설정

3. 한국으로 시간 설정

2. EC2 환경 설정

1. Docker 설치

Docker Compose 설치

* 프리티어 이용 시 RAM 1GB → 2GB로 늘리기

2. Docker Mysql DB

3. Docker Redis

3. Dockerfile로 Jenkins images 받기 (Docker in Docker, DinD 방식)

3. Jenkins 초기 세팅 및 테스트

Jenkins 플러그인 설정

Gitlab

Docker

6. CI/CD (빌드 및 배포) 세팅

빌드 확인

Webhook 설정

배포 환경 구성

도메인 구매

SSL 발급 받기

HTTPS 적용

Front https

Back https

배포 shell 파일

docker-compose-prod.yml

빌드 절차

배포시 주의 사항

DB 정보

Maria DB

Redis + mongoDB

외부 서비스

1. Google 로그인

2. Kakao 로그인

3. Naver 로그인

4. applicatio.yml 설정

OAuth2 설정

SMTP

S3 Bucket

서비스 이용 방법

[Google Cloud Platform](#)
[SMTP - 구글 메일 설정](#)
[S3 버킷](#)
[ELK Stack + Kafka](#)
로그가 수집되는 전체적인 흐름
[Kafka 배포하기](#)
[Filebeat 배포하기](#)
배포 환경
[Spring 서버 설정하기](#)

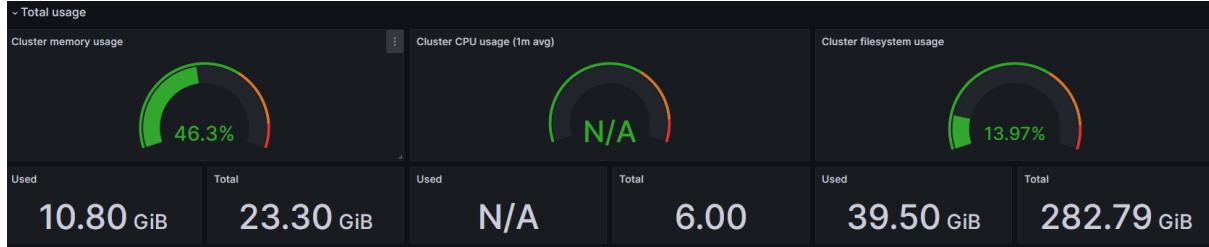
- [1. 라이브러리 추가](#)
- [2. logback 설정 \(application.yml\)](#)
- [3. logback.xml](#)
- [4. Spring 서버 배포하기](#)

[Filebeat 배포하기](#)
[filebeat 폴더 생성](#)
[Dockerfile](#)
[filebeat.yml](#)
[filebeat 실행하기](#)
[ELK 배포하기](#)
배포 환경
[elk 설정 파일을 저장할 디렉토리를 생성하고 이동한다.](#)
[docker compose를 이용하여 한꺼번에 배포할 것이므로, docker compose 설정 파일을 구성한다.](#)
[logstash 관련 설정](#)
[elasticsearch 관련 설정](#)
[kibana 관련 설정](#)
[ELK 실행하기](#)
[로그 수집 확인하기](#)
[elasticsearch에서 확인하기](#)
[kibana에서 확인하기](#)
[MSA 분산 서비스 구축](#)
[spring 서버 및 DB를 기능별로 분리](#)
[Kubernetes 적용](#)
[k8s 수동 설치](#)
[컨테이너 런타임 구성](#)
[kubeadm, kubelet 및 kubectl 설치](#)
[넷필터 브릿지 설정](#)
[클러스터 구성](#)
[마스터 노드 초기화](#)
[유저 설정](#)
[워커 노드 조인하기](#)
[파드 네트워크 배포](#)
[GKE 클러스터 생성하기](#)
[클라우드 셀 활화](#)
[helm chart](#)
[쿠버네티스 배포](#)
[Argo 설치](#)
[설치 확인](#)
[yaml 매니페스트 파일을 활용한 배포](#)
[헬름차트를 활용한 배포](#)
[Argo CD 최종 결과물](#)
[Monitoring : helm chart를 활용한 그라파나, 프로메테우스 배포](#)
[헬름 레파지토리 추가](#)
[그라파나와 프로메테우스 배포](#)
[배포 확인](#)
[프로메테우스 데이터를 그라파나로 가져오기](#)
[대시보드 구성하기](#)
[모니터링 최종 결과물](#)

개발 환경

클러스터 사양

- Node 3개 사용
- CPU : 2GB * 3
- RAM : 8GB * 3
- Storage : 100GB * 3



EC2 서버 인스턴스 사양

- CPU 정보: Intel Xeon(R) Core 4개
- RAM : 16GB
- Disk : 300GB

```
processor : 0 GenuineIntel
vendor_id : Intel
cpu family : 7
model : 79
model name : Intel(R) Xeon(R) CPU E5-2688 v4 @ 2.30GHz
stepping : 0x4000040
microcode : 0x4000040
cpu MHz : 2300.273
cache size : 46080 KB
physical id : 0
siblings : 4
cores : 4
cpu cores : 4
apicid : 0
```

Mem	Total	Used	Free	Shared	Buff/Cache	Available
Mem	100.00	100.00	0.00	0.00	1.00	98.00

Filesystem	Size	Used	Avail	Use%	Mounted on
/dev/root	311G	30G	281G	10%	/

UI / UX

- Figma

형상 관리

- Gitlab
- Github

OS

- Ubuntu 20.04
- Windows 10

이슈 관리

- Jira

Communication

- Mattermost
- Notion
- Webex

Front-end

Node	18.13.0
React	18.2.0
React-router-dom:	6.2.1
Styled-components	5.3.3
Axios	0.25.0

Back-end

java	17
Springboot	3.0.1
jUnit	5
gradle	7.6

IDE

- IntelliJ : 2022.3.1
- Visual Studio Code : 1.75.0

DB

- MySQL : 8.0.34
- Redis : 7.0.8

Infra

- Web Server : Nginx
- Jenkins : 2.388
- Docker : 20.10.23

기타 편의 툴

- WSL2
- Postman
- Termius
- MobaXterm

swagger	3
mariaDB	10.11.1 RC

- Docker-compose : 2.15.1
- Kubernetes : 1.25
- Argo CD
- helm chart: v3

Monitoring

- ELK + filebeat
- Grafana + Prometheus

배포 과정

- EC2 인스턴스는 이미 가지고 있다고 가정하겠습니다.

1. SSAFY EC2(서버) 접속

1. WSL 사용하여 EC2에 SSH 연결

- SSH 접속
 1. 최초 접속 시 권한 요구하면 'yes' 입력

```
# sudo ssh -i [pem키 위치] [접속 계정]@[접속할 도메인]
$ sudo ssh -i k8a806T.pem ubuntu@k8a806.p.ssafy.io
```

2. EC2 편리하게 접속하는 법

- EC2 정보가 담긴 config파일을 만들어 번거롭게 pem와 도메인 경로를 쓰지 않고 접속할 수 있다.
 - ssh 전용 폴더 생성

```
mkdir ~/.ssh
cd ~/.ssh // ssh 폴더 생성 및 이동
cp [로컬 pem 키 위치] ~/.ssh // pem 키 옮기기
vi config // config 파일 생성
```

- config 내용 추가

```
Host ssafy
  HostName [서버 ip 주소]
  User ubuntu
  IdentityFile ~/.ssh/[pem키 파일 명].pem
```

- ssafy 계정에 접속

```
ssh ssafy
```

2. EC2 초기 설정

```
sudo apt update
sudo apt upgrade -y
sudo apt install -y build-essential
```

3. 한국으로 시간 설정

```
sudo ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime  
# 시간 확인  
date
```

```
:~$ date  
Fri May 14 14:12:02 KST 2021
```

2. EC2 환경 설정

1. Docker 설치

1. 기본 설정, 사전 설치

```
sudo apt -y update  
sudo apt install -y apt-transport-https ca-certificates curl software-properties-common
```

2. 자동 설치 스크립트 활용

- 리눅스 배포판 종류를 자동으로 인식하여 Docker 패키지를 설치해주는 스크립트를 제공

```
sudo wget -qO- https://get.docker.com/ | sh
```

3. Docker 서비스 실행하기 및 부팅 시 자동 실행 설정

```
sudo systemctl start docker  
sudo systemctl enable docker
```

4. Docker 그룹에 현재 계정 추가

```
sudo systemctl enable docker  
sudo systemctl restart docker
```

- sudo를 사용하지 않고 docker를 사용할 수 있다.
- docker 그룹은 root 권한과 동일하므로 꼭 필요한 계정만 포함
- 현재 계정에서 로그아웃한 뒤 다시 로그인

5. Docker 설치 확인

```
docker -v
```

```
모든 도커 컨테이너 보기  
docker ps -a  
  
해당 컨테이너의 로그 확인하기  
docker logs [컨테이너ID]  
  
도커 컨테이너 삭제하기  
docker rm -f [컨테이너ID]  
  
도커 이미지 목록 확인하기  
docker images  
  
도거 이미지 삭제하기  
docker rmi -f [이미지ID]  
  
자원 사용량 확인  
ps -eo pid,ppid,cmd,%mem,%cpu --sort=-%mem | head  
  
현재 작동중인 모든 컨테이너의 상위 계층인 machine.slice 에 memoryMax와 memoryHigh 옵션을 설정  
systemctl --runtime set-property machine.slice MemoryHigh=3G MemoryMax=4G  
  
설정 되돌리기  
systemctl revert machine.slice
```

Docker Compose 설치

- 최신 버전을 가져오기 위한 jq 라이브러리 설치

```
$ sudo apt install jq
```

- docker-compose 최신 버전 설치

```
$ VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)
$ DESTINATION=/usr/bin/docker-compose
$ sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-$(uname -s)-$(uname -m) -o $DESTINATION
$ sudo chmod 755 $DESTINATION

// 터미널 재접속 하기!

$ docker-compose -v
Docker Compose version v2.x.x
```

* 프리티어 이용 시 RAM 1GB → 2GB로 늘리기

```
$ sudo dd if=/dev/zero of=/swapfile bs=128M count=16
$ sudo chmod 600 /swapfile
$ sudo mkswap /swapfile
$ sudo swapon /swapfile
$ sudo swapon -s

$ sudo vi /etc/fstab
/swapfile swap swap defaults 0 0 // 제일 아랫줄에 추가 ! (이 주석은 빼고)
```

2. Docker Mysql DB

```
docker pull mysql
```

mysql 도커 실행

```
docker run -p 3306:3306 --name user-db -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=ssafy806_user -e MYSQL_CHARSET=utf8 -e
MYSQL_COLLATION=utf8_general_ci -d mysql
docker run -p 3307:3306 --name payment-db -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=ssafy806_payment -e MYSQL_CHARSET=utf8 -e
MYSQL_COLLATION=utf8_general_ci -d mysql
docker run -p 3308:3306 --name concert-db -e MYSQL_ROOT_PASSWORD=root -e MYSQL_DATABASE=ssafy806_concert -e MYSQL_CHARSET=utf8 -e
MYSQL_COLLATION=utf8_general_ci -d mysql
```

mysql 컨테이너 접속

```
docker exec -it db-user /bin/bash
docker exec -it db-payment /bin/bash
docker exec -it db-concert /bin/bash
```

root 계정 접속 (비밀번호 : root)

```
mysql -u root -p
```

아이디 생성

```
create user '[아이디]'@'[호스트]' identified with mysql_native_password by '[비밀번호]';
```

예 :

```
create user 'ssafy806'@'%' identified with mysql_native_password by 'ssafy806';
```

변경 사항 적용

```
flush privileges;
```

사용할 DB 생성

```
create database ssafy806_user;
create database ssafy806_payment;
create database ssafy806_concert;
```

root 계정 접속

```
grant all privileges on [db스키마].[권한] to '[아이디]'@'[호스트]';
```

예 :

```
grant all privileges on ssafy806_user.* to 'ssafy806'@'%';
```

변경 사항 적용

```
flush privileges;
```

3. Docker Redis

- Redis 이미지 받기

```
docker pull redis:alpine
```

- 도커 네트워크 생성 [디폴트값]

```
docker network create redis-network
```

- 도커 네트워크 상세정보 확인

```
docker inspect redis-network
```

- local-redis라는 이름으로 로컬-docker 간 6379 포트 개방

```
docker run --name local-redis -p 6379:6379 --network redis-network -v /redis_temp:/data -d redis:alpine redis-server --appendonly yes
```

- Docker 컨테이너 확인

```
docker ps -a
```

- 컨테이너 진입

```
# 실행 중인 redis 컨테이너에 대해 docker redis-cli 로 직접 진입
docker run -it --network redis-network --rm redis:alpine redis-cli -h local-redis

# bash로도 진입 가능하다.
docker run -it --network redis-network --rm redis:alpine bash
redis-cli
```

- 권한 추가

```
# slaveof no one : 현재 슬레이브(복제)인 자신을 마스터로 만듭니다.
127.0.0.1:6379> slaveof no one
```

- 테스트

- OK 가 뜨면 성공

```

127.0.0.1:6379> slaveof no one
OK
127.0.0.1:6379> set apple 100
OK
127.0.0.1:6379> get apple
"100"

```

3. Dockerfile로 Jenkins images 받기 (Docker in Docker, DiD 방식)

- Dockerfile 작성

```

# 폴더 생성
mkdir config && cd config

# 아래 내용 작성
$ vi Dockerfile

FROM jenkins/jenkins:jdk17

#도커를 실행하기 위한 root 계정으로 전환
USER root

#도커 설치
COPY docker_install.sh /docker_install.sh
RUN chmod +x /docker_install.sh
RUN /docker_install.sh

#설치 후 도커그룹의 jenkins 계정 생성 후 해당 계정으로 변경
RUN groupadd -f docker
RUN usermod -aG docker jenkins
USER jenkins

```

- docker 설치 shell 파일

```

#!/bin/sh
apt-get update && \
apt-get -y install apt-transport-https \
ca-certificates \
curl \
gnupg2 \
zip \
unzip \
software-properties-common && \
curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey && \
add-apt-repository \
"deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") \
$(lsb_release -cs) \
stable" && \
apt-get update && \
apt-get -y install docker-ce

```

- Docker 이미지 생성

```
docker build -t jenkins/myjenkins .
```

- Docker 볼륨 폴더 권한 설정

```
$ mkdir /var/jenkinsDir/
$ sudo chown 1000 /var/jenkinsDir/
```

- Jenkins 컨테이너 생성

```

docker run -d -p 9090:8080 --name=jenkinsci \
-e TZ=Asia/Seoul \
-v /var/jenkinsDir:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
jenkins/myjenkins

```

- 옵션 설명

`-d` : 는 백그라운드에서 실행을 의미

`-p` 는 매핑할 포트를 의미합니다. (p가 port의 단축어가 아니었음 ..)

`:` 기준으로 왼쪽은 로컬포트, 오른쪽은 도커 이미지의 포트를 의미합니다. 도커 이미지에서의 8080 포트를 로컬 포트 9090으로 매핑한다는 뜻입니다.

```
-v /var/run/docker.sock:/var/run/docker.sock \
jenkins/myjenkins
```

이 옵션은 로컬의 도커와 젠킨스 내에서 사용할 도커 엔진을 동일한 것으로 사용하겠다는 의미입니다.

`v` 옵션은 ":"를 기준으로 왼쪽의 로컬 경로를 오른쪽의 컨테이너 경로로 마운트 해줍니다.

즉, 제 컴퓨터의 사용자경로/jenkinsDir 을 컨테이너의 /var/jenkins_home 과 바인드 시켜준다는 것입니다. 물론, 양방향으로 연결됩니다.

컨테이너가 종료되거나 알 수 없는 오류로 정지되어도, jenkins_home에 남아있는 소중한 설정 파일들은 로컬 경로에 남아있게 됩니다.

3. Jenkins 초기 세팅 및 테스트

- 젠킨스에 접속하기 전에 `/var/run/docker.sock` 에 대한 권한을 설정해주어야 합니다.
- 초기 `/var/run/docker.sock` 의 권한이 소유자와 그룹 모두 root였기 때문에 이제 그룹을 root에서 `docker` 로 변경해줄겁니다.
- 먼저, jenkins로 실행됐던 컨테이너의 bash를 root 계정으로 로그인 하기전에, 현재 실행되고 있는 컨테이너의 정보들을 확인할 수 있는 명령어를 입력해 아이디를 확인하겠습니다.

```
docker ps -a
```

```
> docker ps -a
CONTAINER ID        IMAGE               COMMAND             CREATED            STATUS              PORTS               NAMES
0bcd8291015        jenkins/myjenkins   "/sbin/tini -- /usr/..."   23 seconds ago   Up 21 seconds          5000/tcp, 0.0.0.0:9090->8080/tcp   jenkinsciid
```

- 우리가 방금 생성한 컨테이너의 ID는 **0bcd8291015**입니다. 도커는 다른 컨테이너 ID와 겹치지 않는 부분까지 입력하면 해당 컨테이너로 알아서 매핑해줍니다.

```
docker exec -it -u root 컨테이너ID /bin/bash
```

`exec` 는 컨테이너에 명령어를 실행시키는 명령어인데, `/bin/bash`와 옵션 `-it`를 줌으로써 컨테이너의 헬에 접속할 수 있습니다.

이제 정말로 root 계정으로 컨테이너에 접속하기 위해 컨테이너ID에 0bc를 입력해 실행합니다.

```
> docker exec -it -u root 0bc /bin/bash
root@0bcd8291015:/#
```

- root 계정으로 로그인이 잘 되었습니다. 이제 그룹을 바꾸기 위해 다음 명령어를 실행해줍니다.

```
chown root:docker /var/run/docker.sock
```

- 그리고 이제 헬을 exit 명령어로 빠져나온 후 다음 명령어를 실행해 컨테이너를 재실행해줍니다.

```
docker restart [컨테이너 ID]
```

- Jenkins 패스워드 확인

```
docker logs [jenkins 컨테이너 ID]
```

- docker logs 컨테이너 id를 입력해 로그를 출력하면 initialAdminPassword가 출력됩니다. 이 패스워드를 입력해주면 됩니다.

```
*****
Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:
b99f67a2cf054174a73017e4498ce87d
This may also be found at: /var/jenkins_home/secrets/initialAdminPassword
*****
*****
```

- 보안 그룹 설정을 해야 {public ip}:9090에 접근할 수 있습니다.

EC2 > 보안 그룹 >

인바운드 규칙 (2)

인바운드 규칙 편집 클릭

인바운드 규칙	정보
sgr-04d3d90131b6c0b79	유형: 사용자 지정 TCP 프로토콜: TCP 포트 범위: 9090 소스: 사용자 지정 설명: 선택 사항 상태: 삭제
sgr-05db5853e7a0e7cbf	유형: SSH 프로토콜: TCP 포트 범위: 22 소스: 사용자 지정 설명: 선택 사항 상태: 삭제

규칙 추가

규칙 추가 클릭 → 유형: TCP, 포트 범위: {등록할 포트 번호}

Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

Install suggested plugins

Install plugins the Jenkins community finds most useful.

Select plugins to install

Select and install plugins most suitable for your needs.

- 정상적으로 입력했다면 플러그인 설치가 나오는데, 우리는 Install suggested plugins를 선택합니다.

<input checked="" type="checkbox"/> Folders	<input type="checkbox"/> OWASP Markup Formatter	<input type="checkbox"/> Build Timeout	<input type="checkbox"/> Credentials Binding	** SSH server Folders ** Trilead API
<input type="checkbox"/> Timestamper	<input type="checkbox"/> Workspace Cleanup	<input type="checkbox"/> Ant	<input type="checkbox"/> Gradle	
<input type="checkbox"/> Pipeline	<input type="checkbox"/> GitHub Branch Source	<input type="checkbox"/> Pipeline: GitHub Groovy Libraries	<input type="checkbox"/> Pipeline: Stage View	
<input type="checkbox"/> Git	<input type="checkbox"/> SSH Build Agents	<input type="checkbox"/> Matrix Authorization Strategy	<input type="checkbox"/> PAM Authentication	
<input type="checkbox"/> LDAP	<input type="checkbox"/> Email Extension	<input type="checkbox"/> Mailer		

- 설치가 완료되면, 어드민 계정 생성창이 나오고, 본인이 사용하실 정보들을 입력해줍시다.

Create First Admin User

계정명:	<input type="text"/>
암호:	<input type="password"/>
암호 확인:	<input type="password"/>
이름:	<input type="text"/>
이메일 주소:	<input type="text"/>

Instance Configuration

Jenkins URL:

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the `BUILD_URL` environment variable provided to build steps.

The proposed default value shown is **not saved yet** and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

- 앞으로 이 url로 젠킨스에 접속하시면 됩니다.

Jenkins 플러그인 설정

- Gitlab, Docker 플러그인을 받습니다.

Gitlab

The screenshot shows the Jenkins plugin search interface. A search bar at the top contains the text "Gitlab". Below the search bar, a list of plugins is displayed. The first plugin listed is "Generic Webhook Trigger Plugin" version 1.86.2, which can receive any HTTP request, extract values from JSON, and more. It includes a link to "Report an issue with this plugin". The second plugin listed is "GitLab" version 1.6.0, which allows Jenkins to trigger builds and display GitLab status. It also includes a link to "Report an issue with this plugin". The third plugin listed is "Gitlab API Plugin" version 5.0.1-78.v47a_45b_9f78b_7, which provides GitLab API for other plugins. It includes a link to "Report an issue with this plugin". The fourth plugin listed is "GitLab Authentication plugin" version 1.16, which is an authentication plugin using gitlab OAuth. It includes a link to "Report an issue with this plugin". At the bottom of the list, there is a yellow box containing the text "This plugin is up for adoption! We are looking for new maintainers. If you're interested, please let us know via GitHub or our community forum.".

Q Gitlab

이름 ↓

[Generic Webhook Trigger Plugin](#) 1.86.2
Can receive any HTTP request, extract any values from JSON and many more.
[Report an issue with this plugin](#)

[GitLab](#) 1.6.0
This plugin allows [GitLab](#) to trigger Jenkins builds and display GitLab status.
[Report an issue with this plugin](#)

[Gitlab API Plugin](#) 5.0.1-78.v47a_45b_9f78b_7
This plugin provides [GitLab API](#) for other plugins.
[Report an issue with this plugin](#)

[GitLab Authentication plugin](#) 1.16
This is the an authentication plugin using gitlab OAuth.
[Report an issue with this plugin](#)

This plugin is up for adoption! We are looking for new maintainers. If you're interested, please let us know via GitHub or our community forum.

Docker

The screenshot shows the Jenkins plugin store interface. A search bar at the top contains the text 'Docker'. Below it, a list of plugins is displayed, each with a title, version, a brief description, and a 'Report an issue with this plugin' link. Some descriptions include a note that the plugin is up for adoption. The plugins listed are:

- Docker API Plugin** 3.2.13-37.vf3411c9828b9
This plugin provides **docker-java** API for other plug
[Report an issue with this plugin](#)
This plugin is up for adoption! We are looking for maintainers.
- Docker Commons Plugin** 1.21
Provides the common shared functionality for various Docker related Jenkins Plugins.
[Report an issue with this plugin](#)
- Docker Compose Build Step Plugin** 1.0
Docker Compose plugin for Jenkins
[Report an issue with this plugin](#)
- Docker Pipeline** 563.vd5d2e5c4007f
Build and use Docker containers from pipelines.
[Report an issue with this plugin](#)
This plugin is up for adoption! We are looking for maintainers.
- Docker plugin** 1.3.0
This plugin integrates Jenkins with **Docker**
[Report an issue with this plugin](#)
This plugin is up for adoption! We are looking for maintainers.
- docker-build-step** 2.9
This plugin allows to add various docker commands as build steps.
[Report an issue with this plugin](#)

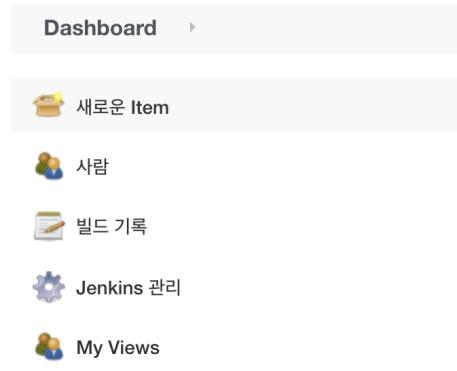
Jenkins is ready!

Your Jenkins setup is complete.

[Start using Jenkins](#)

- 여기까지 오셨다면, 젠킨스 설치 및 초기 세팅 완료!

6. CI/CD (빌드 및 배포) 세팅



- 먼저, 대쉬보드의 새로운 아이템을 클릭합니다.

Enter an item name

* Required field

Freestyle project

이것은 Jenkins의 주요 기능입니다. Jenkins는 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.

- 아이템 이름을 자유롭게 입력해주시고, Freestyle project를 선택하고, OK로 생성합니다.
- 이제 빌드 설정창이 뜰텐데, 소스 코드 관리쪽에서 Git을 선택하고, Repository URL에 다음과 같이 입력해줍니다.
- Gitlab 저장소를 입력하시면 됩니다.

소스 코드 관리

None
 Git

Repositories

Repository URL

Credentials

- Credentials에서 Username / password로 선택하고 SSAFY Email로 두가지 모두 입력하고 계정 인증을 진행합니다.
- 이제 Build에서 Execute shell을 선택해줍니다.

Build

Add build step ▲

Execute Windows batch command

Execute shell

- 폴더 내에 prod와 dev 두가지 버전이 존재하며 prod 환경 기준으로 설명하겠습니다.
- Gitlab의 저장소와 연동되어 폴더 내의 start-prod.sh 파일이 실행 됩니다.

```
bash start-prod.sh
```

빌드 확인

- 이제 드디어 세팅한 값들을 확인해볼 차례입니다. 위의 내용들을 저장하고 Build Now를 눌러봅니다.

Webhook 설정

- Jenkins 트리거 체크

Build when a change is pushed to GitLab. GitLab webhook URL: http://i8a60...

- Jenkins에서 **빌드 유발** → **Build when ...** → **고급** → 하단에 **Secret token Generate** → 토큰 발급 완료!
- Gitlab Webhook에서 해당 토큰을 등록합니다.
 1. lab.ssafy.com Gitlab 프로젝트 접속
 2. 좌측 Settings → Webhook 접속
 3. 아래와 같이 URL 란에 Jenkins에서의 Item URL를 입력하고 Save를 눌러줍니다.

Build when a change is pushed to GitLab. GitLab webhook URL: http://ssafy.io: /project/shabit-main ?

URL
http://example.com/trigger-ci.json
URL must be percent-encoded if it contains one or more special characters.

Secret token

Used to validate received payloads. Sent with the request in the X-Gitlab-Token HTTP header.

Trigger
 Push events
Branch name or wildcard pattern to trigger on (leave blank for all)

배포 환경 구성

도메인 구매

- 가비아 접속

웹을 넘어 클라우드로. 가비아
그룹웨어부터 멀티클라우드까지 하나의 클라우드 허브

q. <https://www.gabia.com/>

- 도메인 선택
!!아래는 예시이고 실제 구매한 도메인은 “shabit.site”입니다.



- 도메인 결제

- DNS 설정
 - My 가비아 → DNS 관리를 → DNS 관리에서 호스트 / 값 설정
 - 호스트에는 @, 값/위치에는 domain 주소를 작성합니다.

SSL 발급 받기

- <https://www.sslforfree.com/> 접속
- 도메인 입력

- 회원가입 및 로그인

Sign Up

Sign up for a free account to create and manage SSL certificates.

Domain: **shabit.site**

Email Address

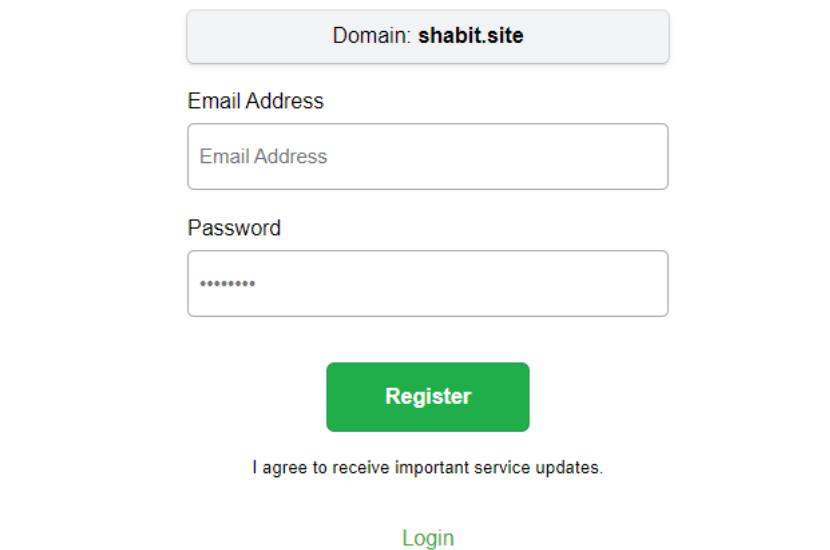
Email Address

Password

Register

I agree to receive important service updates.

[Login](#)

A screenshot of a sign-up form for creating and managing SSL certificates. It includes fields for domain name, email address, password, and a registration button. There is also a checkbox for agreeing to receive service updates and a link to log in.

- SSL 발급 셋팅

SSL Certificate Setup

You're on your way to issuing a brand-new SSL certificate for one or multiple domains. Before you can install your new certificate, please complete the steps below.

Domains

I need a wildcard certificate PRO

Please enter at least one domain to secure. For single-domain certificates the WWW-version of your domain will always be included at no extra charge.

Enter Domains

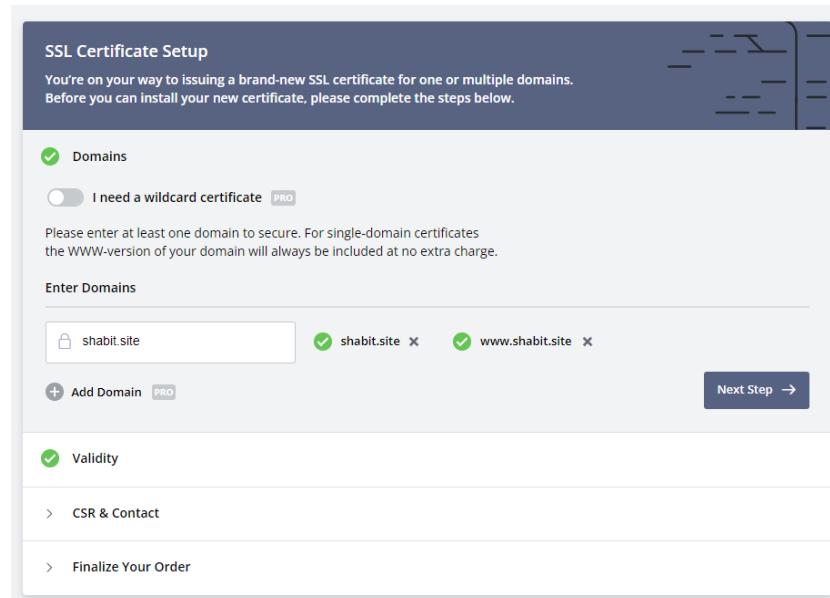
shabit.site shabit.site www.shabit.site

[+ Add Domain PRO](#) [Next Step →](#)

Validity

> CSR & Contact

> Finalize Your Order

A screenshot of the SSL Certificate Setup interface. It shows the user has selected two domains: 'shabit.site' and 'www.shabit.site'. The 'Domains' section is marked as completed with a green checkmark. The 'Validity' section is also marked as completed. Navigation links for 'CSR & Contact' and 'Finalize Your Order' are visible at the bottom.

SSL Certificate Setup

You're on your way to issuing a brand-new SSL certificate for one or multiple domains. Before you can install your new certificate, please complete the steps below.

Domains

Validity

You can now choose between generating 90-day or one-year certificate validity. To keep manual work at a minimum, we recommend 1-year certificates.

90-Day Certificate

1-Year Certificate PRO

Next Step →

CSR & Contact

> Finalize Your Order

Domains

Validity

CSR & Contact

Finalize Your Order

Based on your selection of a **90-Day SSL Certificate** you are fine staying on the **Free Plan**. To create and validate your SSL Certificate, please click "Next Step" below.

Free \$0 / month <input checked="" type="checkbox"/> Selected	Basic \$10 / month or \$8 if billed yearly <input type="checkbox"/> Select	Premium \$50 / month or \$40 if billed yearly <input type="checkbox"/> Select	Business \$100 / month or \$80 if billed yearly <input type="checkbox"/> Select
<ul style="list-style-type: none"> <input checked="" type="checkbox"/> 3 90-Day Certificates <input type="checkbox"/> 1-Year Certificates <input type="checkbox"/> Multi-Domain Certs <input type="checkbox"/> 90-Day Wildcards <input type="checkbox"/> 1-Year Wildcards <input type="checkbox"/> REST API Access <input type="checkbox"/> Technical Support 	<ul style="list-style-type: none"> <input type="checkbox"/> 90-Day Certificates <input checked="" type="checkbox"/> 3 1-Year Certificates <input checked="" type="checkbox"/> Multi-Domain Certs <input type="checkbox"/> 90-Day Wildcards <input type="checkbox"/> 1-Year Wildcards <input checked="" type="checkbox"/> REST API Access <input checked="" type="checkbox"/> Technical Support 	<ul style="list-style-type: none"> <input type="checkbox"/> 90-Day Certificates <input checked="" type="checkbox"/> 10 1-Year Certificates <input checked="" type="checkbox"/> Multi-Domain Certs <input type="checkbox"/> 90-Day Wildcards <input checked="" type="checkbox"/> 1 1-Year Wildcards <input checked="" type="checkbox"/> REST API Access <input checked="" type="checkbox"/> Technical Support 	<ul style="list-style-type: none"> <input type="checkbox"/> 90-Day Certificates <input checked="" type="checkbox"/> 25 1-Year Certificates <input checked="" type="checkbox"/> Multi-Domain Certs <input type="checkbox"/> 90-Day Wildcards <input checked="" type="checkbox"/> 3 1-Year Wildcards <input checked="" type="checkbox"/> REST API Access <input checked="" type="checkbox"/> Technical Support

- SSL 인증서를 받을 때 google.com 같은 사이트의 인증서 발급을 막기 위해서 도메인 인증을 해야합니다.

shabit.site

Congratulations, your SSL certificate is en route! However, you need to verify ownership of your domain before installing your certificate. Please follow the steps below.

Verification Method for shabit.site

We need you to verify ownership of each domain in your certificate. Please select your preferred verification method and click "Next Step".

Email Verification
 DNS (CNAME)

Follow the steps below

To verify your domain using a CNAME record, please follow the steps below:

- 1 Sign in to your DNS provider, typically the registrar of your domain.
- 2 Navigate to the section where DNS records are managed.
- 3 Add the following CNAME record:

Name: _8C€

Point To: 9669C6B165E260AE2D1
fc.comodoca.com

TTL: 3600 (or lower)

- 4 Save your CNAME record and click "Next Step" to continue.

HTTP File Upload

Next Step →

- 위에서 **Name** 과 **Point To**의 값을 가비아 DNS 관리툴에서 **호스트** / **값**에 추가 해줍니다.

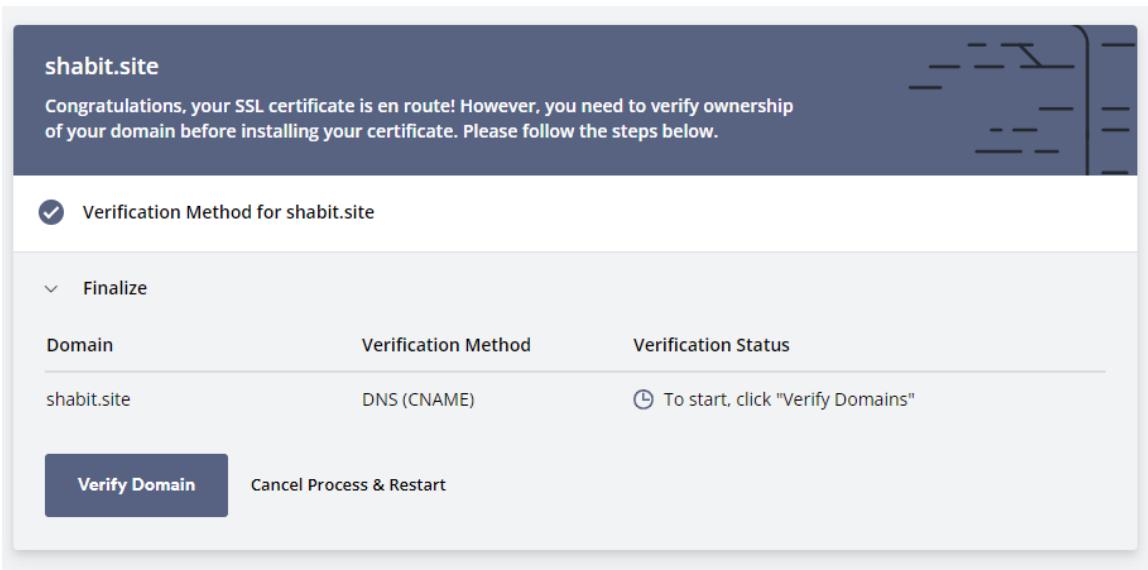
DNS 관리

shabit.site

• 레코드 개수 : 3개 • 최근 업데이트 : 2023-02-07 22:36:25 • 네임서버 : ns.gabia.co.kr

타입	호스트	값/위치
CNAME	@	l8a601.p.ssafy.io.
CNAME	_0fa955987da3d abb889	9c2fe23e41d84ee0e3e20628efe1ef90.e21fd6610e06a
CNAME	www	l8a601.p.ssafy.io.

- 인증 후 인증서 압축 파일을 발급받습니다.



HTTPS 적용

Front https

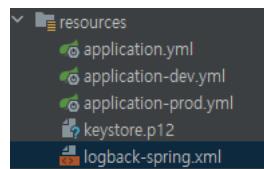
- Verify Domain 버튼을 눌러 도메인을 인증
- 성공한다면, Server Type을 Nginx로 선택한 후, 인증서를 다운로드 받습니다.
- 압축을 풀고 /.nginx/cert 폴더에 저장합니다.

Back https

- 백엔드에서 Https를 적용하기 위해서는 인증서를 pem키로 변환 해주어야 합니다.

```
sudo openssl pkcs12 -export -out keystore.p12 -inkey private.key -in certificate.crt -certfile ca_bundle.crt
```

- 이후 생성된 keystore.p12 파일을 resources에 추가합니다.



- yaml 파일에 ssl 설정 값을 추가합니다.

```
server:
  ssl:
    key-store: classpath:keystore.p12
    key-store-password: ssafy
    key-store-type: PKCS12
```

배포 shell 파일

```
docker-compose -f docker-compose-prod.yml pull
COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose-prod.yml up --build -d
docker rmi -f $(docker images -f "dangling=true" -q) || true
```

docker-compose-prod.yml

```

version: "3"
services:
  server:
    container_name: server
    build:
      context: ./SHabit-back
      args:
        SERVER_MODE: prod
    ports:
      - 8080:8080
    environment:
      - TZ=Asia/Seoul
  client:
    container_name: client
    build:
      context: ./shabit-front
      dockerfile: Dockerfile.dev
    ports:
      - 3000:3000
    depends_on:
      - server
  nginx:
    container_name: nginx
    build: ./nginx
    depends_on:
      - server
      - client
    volumes:
      - .nginx/conf.d:/etc/nginx/conf.d
      - .nginx/zeroSSL:/var/www/zeroSSL/.well-known/pki-validation
      - .nginx/cert:/cert
    ports:
      - 80:80
      - 443:443

```

빌드 절차

- Gitalb에서 Jenkins로 Webhooks을 연동한 다음 해당 브랜치에 push, merge 를 진행합니다.
 - start-prod.sh 쉘 파일 실행
 - docker-compose 실행
 - Dockerfile 실행
 - Server → Front 순으로 배포 진행

배포시 주의 사항

1. 프론트 엔드에서 서버 주소는 shabit.site 이므로 로컬 환경에서 구동할 시 localhost로 변경해야합니다.
2. 백엔드는 prod, dev, local 세 가지 파일로 분리하여 ip 주소는 건들이지 않고 db 주소 및 계정 정보를 변경하면 됩니다.

DB 정보

Maria DB

```

datasource:
  driver-class-name: org.mariadb.jdbc.Driver
  url: jdbc:mariadb://localhost:3306/ssafy
  username: easypeasy
  password: lemon

```

Redis + mongoDB

```
data:  
  redis:  
    host: localhost  
    port: 6379  
  
  mongodb:  
    host: localhost  
    port: 27017  
    authentication-database: admin  
    username: root  
    password: root  
    database: ssafy
```

외부 서비스

1. Google 로그인

1. Google Cloud 접속

Cloud Computing Services | Google Cloud

Meet your business challenges head on with cloud computing services from Google, including data management, hybrid & multi-cloud, and AI & ML.

 <https://cloud.google.com/>

2. 최초 가입이라면 카드 정보 등록
3. API 및 서비스
4. 프로젝트 생성
5. 사용자 인증 정보 - OAuth2 클라이언트 ID - 웹 어플리케이션 선택

API API 및 서비스	사용자 인증 정보	 사용자 인증 정보 만들기	삭제				
<ul style="list-style-type: none">❖ 사용 설정된 API 및 서비스■ 라이브러리● 사용자 인증 정보▷ OAuth 동의 화면≡ 페이지 사용 동의	<p>사용 설정한 API에 액세스하려면 사용자 인증 정보를 만드세요. 자세히 알아보기.</p> <p>API 키</p> <table border="1"><thead><tr><th>□ 이름</th><th>생성일</th></tr></thead><tbody><tr><td>표시할 API 키가 없습니다.</td><td></td></tr></tbody></table> <p>OAuth 2.0 클라이언트 ID</p>	□ 이름	생성일	표시할 API 키가 없습니다.			
□ 이름	생성일						
표시할 API 키가 없습니다.							

1. 서버 URI 추가

- ex) <https://shabit.site/login/oauth2/code/google>

승인된 리디렉션 URI ?

웹 서버의 요청에 사용

URI 1 *

! 올바르지 않은 리디렉션 URI는 비워 둘 수 없습니다.

[+ URI 추가](#)

참고: 설정이 적용되는 데 5분에서 몇 시간이 걸릴 수 있습니다.

[만들기](#) [취소](#)

- 생성 후 클라이언트 ID 및 Secret 키 저장



2. Kakao 로그인

- 카카오 개발자 사이트 접속

Kakao Developers
카카오 API를 활용하여 다양한 어플리케이션을 개발해보세요. 카카오 로그인, 메시지 보내기, 친구 API, 인공지능 API 등을 제공합니다.

<https://developers.kakao.com/>

kakao developers

- 새 애플리케이션 생성
- 좌측 제품 설정 → 카카오 로그인
- 활성화 설정
- 하단 Redirect URI에 서버 주소 추가
 - <https://shabit.site/login/oauth2/code/kakao>
- 카카오 이메일을 필수 항목으로 지정할 경우 비즈니스 앱 등록을 해야한다.

동의항목

카카오 로그인으로 서비스를 시작할 때 동의 받는 항목을 설정
비즈니스 채널을 연결하면 권한이 없는 동의 항목에 대한 검수

카카오비즈니스 관리자센터에서 비즈니스 채널 연결 →

7. 클릭 후 비즈 - 앱 연결 확인 후 목적은 **[이메일 필수 항목]** 으로 설정

8. 카카오 클라이언트 ID ⇒ 앱키 - REST API 키

앱 키

플랫폼	앱 키
네이티브 앱 키	5b5f7
REST API 키	7b4cc
JavaScript 키	8555e
Admin 키	36c4c

9. 카카오 클라이언트 Secret 키 : 제품 설정 → 보안 → 키 발급

Client Secret

토肯 발급 시, 보안을 강화하기 위해 Client

코드	yj34MVM
활성화 상태	사용안함

- 활성화 상태는 '사용안함'로 설정!

3. Naver 로그인

- Naver 로그인은 앱 개발자 등록 및 해당 검수 요청을 신청해 인증을 받아야만 사용할 수 있다.
- 네이버 개발자 사이트 접속

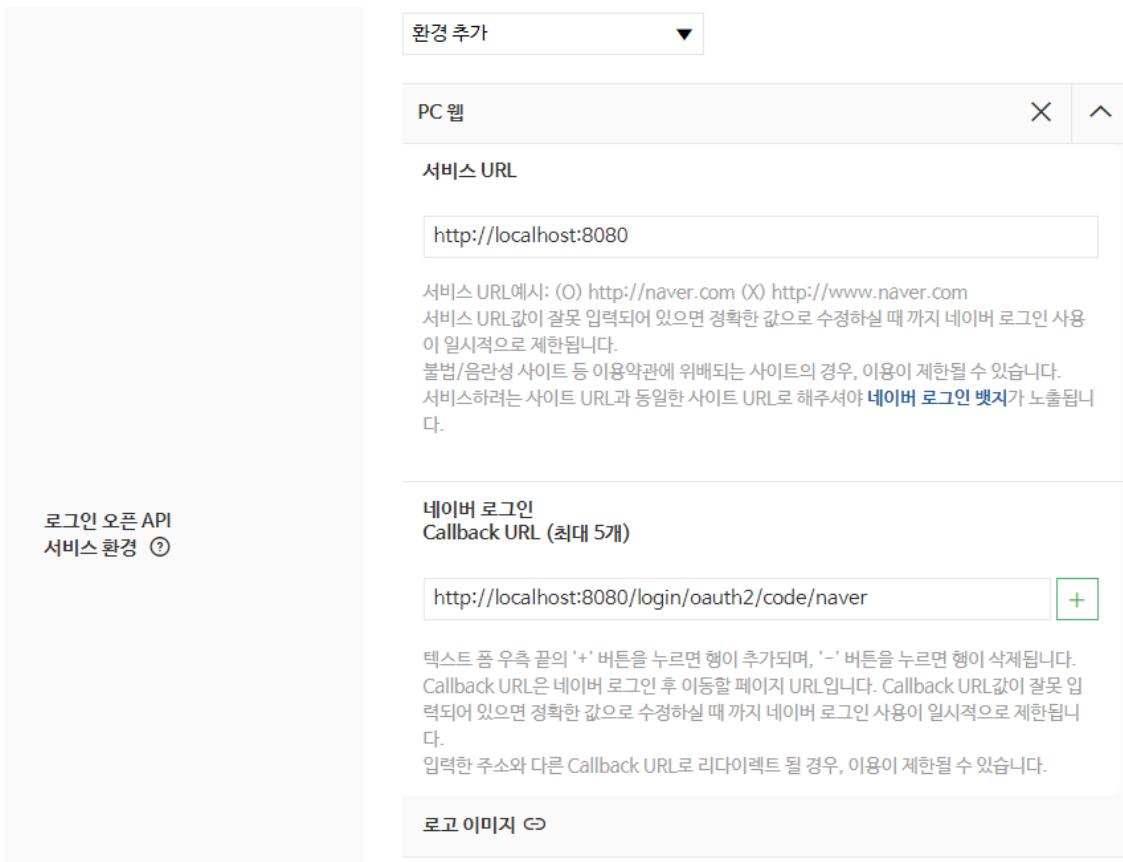
NAVER Developers

네이버 오픈 API들을 활용해 개발자들이 다양한 애플리케이션을 개발할 수 있도록 API 가이드와 SDK를 제공합니다.
제공중인 오픈 API에는 네이버 로그인, 검색, 단축URL, 캡차를 비롯 기계번역, 음성인식, 음성합성 등이 있습니다.

<https://developers.naver.com/main/>



- 상단 Application → 내 애플리케이션 접속
- 프로젝트 선택
- API 설정
- 로그인 오픈 API서비스 환경 설정



6. 앱 로고 등록 후 프로젝트 템 상단에 네이버 로그인 검수 상태 신청
7. 요구하는 이미지를 등록하고 검수 요청 하면 1~2일 이내로 완료됩니다.

<input checked="" type="checkbox"/> 검수 신청 전 가이드 확인	<input checked="" type="checkbox"/> 검수 요청 가이드 확인
--	--

아래 정보들은 [API 설정] 탭에서 조회할 정보로 선택하신 항목입니다.

네이버 로그인을 적용하려는 서비스 페이지에서 각각의 항목이 보여지는 화면을 캡처하여 업로드하고, 첨부 여부 체크박스를 체크하여 모든 항목이 업로드 되었는지 다시 한번 확인해 주세요.

만약 사용하지 않는 정보가 있다면 [API 설정] 탭으로 이동하여 해당 항목의 체크박스를 해제하고 수정 내역을 저장해 주세요.

① 제공 정보 활용처 확인 ↗	<table border="1" style="width: 100%; border-collapse: collapse;"> <thead> <tr> <th style="width: 30%;">제공 정보(권한)</th> <th style="width: 30%;">필수/추가 여부</th> <th style="width: 40%;">첨부 여부</th> </tr> </thead> <tbody> <tr> <td style="padding: 5px;">이메일 주소</td> <td style="padding: 5px;">필수</td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> <tr> <td style="padding: 5px;">회원이름</td> <td style="padding: 5px;">필수</td> <td style="padding: 5px;"><input type="checkbox"/></td> </tr> </tbody> </table>	제공 정보(권한)	필수/추가 여부	첨부 여부	이메일 주소	필수	<input type="checkbox"/>	회원이름	필수	<input type="checkbox"/>
제공 정보(권한)	필수/추가 여부	첨부 여부								
이메일 주소	필수	<input type="checkbox"/>								
회원이름	필수	<input type="checkbox"/>								

캡처 파일 업로드 ①

- pdf, jpg, gif, png 및 오피스 파일 형식을 첨부하실 수 있습니다.
- 5MB 이하의 파일을 최대 5개까지 업로드 할 수 있습니다.

네이버 로그인을 통한 신규 회원가입에 적용
 회원가입 없이 단순 로그인 인증에 적용
 게시판 글쓰기 또는 댓글 작성 등 일부 메뉴에 적용
 기타

- “카페24” 호스팅사를 통해 사이트를 만드시는 경우, 기타에 체크하고 “카페24”를 적어주세요.
- 네이버 로그인을 신규 회원가입에 적용하실 경우, 네이버 로그인을 통한 신규 회원가입 과정에서 별도의 비밀번호를 요구한다면 검수는 거부됩니다. 관련하여 자세한 사항은 [네이버 로그인 검수 가이드](#)를 참고해 주세요.

② 서비스 적용 형태 확인 ↗

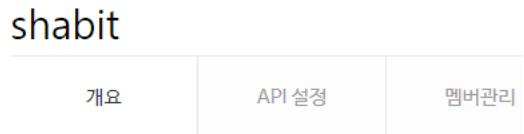
네이버 로그인 이용 절차를 확인할 수 있는 캡처파일 첨부

- 서비스에서 “네이버 로그인” 버튼을 클릭하여 로그인/회원가입을 완료하기까지 노출되는 화면을 단계별로 캡처하여 첨부해 주세요.
- 자세한 내용은 [\[검수 가이드 - 네이버 로그인 적용 형태 확인을 위한 자료 제출 방법\]](#)을 참고해 주세요.
- pdf, jpg, gif, png 및 오피스 파일 형식을 첨부하실 수 있습니다.
- 5MB 이하의 파일을 최대 5개까지 업로드 할 수 있습니다.

아래 내용 중 하나를 선택해 주세요. 해당되는 것이 없으면 “없음”으로 선택해 주세요.

스포츠 게임 배팅, 투표권발행 유사행위를 제공하는 서비스
 온라인으로 주류를 판매하는 서비스

1. 프로젝트 상단에 개요 탭에서 클라이언트 ID, Secret 키 확인 가능



4. application.yml 설정

OAuth2 설정

```
spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: 구글 Client ID
            client-secret: 구글 Client Secret
            redirect-uri: '{baseUrl}/{action}/oauth2/code/{registrationId}'
            scope: profile, email // 사용할 필수 정보들
          naver:
            client-id: 네이버 Client ID
            client-secret: 네이버 Client Secret
            redirect-uri: '{baseUrl}/{action}/oauth2/code/{registrationId}'
            authorization-grant-type: authorization_code
            scope:
              - name
              - email
              - profile_image
            client-name: Naver
          kakao:
            client-id: 카카오 Client ID
            client-secret: 카카오 Client Secret
            scope:
              - profile.nickname
              - account_email
              - profile.image
            client-name: Kakao
            authorization-grant-type: authorization_code
            redirect-uri: '{baseUrl}/{action}/oauth2/code/{registrationId}'
            client-authentication-method: POST

        provider:
          naver:
            authorization-uri: https://nid.naver.com/oauth2.0/authorize
            token-uri: https://nid.naver.com/oauth2.0/token
            user-info-uri: https://openapi.naver.com/v1/nid/me
            user-name-attribut: response
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribut: id

      app: // 클라이언트 인가 코드
        oauth2:
          authorizedRedirectUris: https://shabit.site/oauth/redirect
```

SMTP

```

spring:
  mail:
    host: smtp.gmail.com
    port: 587
    username: {google 메일 아이디}
    password: {google 계정 앱 key}
    properties:
      mail:
        smtp:
          auth: true
          starttls:
            enable: true

  mail:
    setFrom: {google 이메일}

```

S3 Bucket

```

cloud:
  aws:
    s3:
      bucket: 버킷 이름
    region:
      static: ap-northeast-2 #Asia Pacific -> seoul
    stack:
      auto: false
    credentials:
      access-key: S3 사용자 access-key
      secret-key: S3 사용자 secret-key

```

서비스 이용 방법

Google Cloud Platform

1. Google Cloud Platform 접속 후 새 프로젝트 만들기 → 좌측 햄버거 메뉴에서 API 및 서비스 클릭 → 라이브러리 클릭
 2. YouTube Data API v3 검색 → 사용 클릭 → 관리 클릭 → 사용자 인증 정보 클릭
 3. 사용자 인증 정보 만들기 → API 키 클릭
2. 발급받은 API 키를 application.yml에 다음과 같이 입력

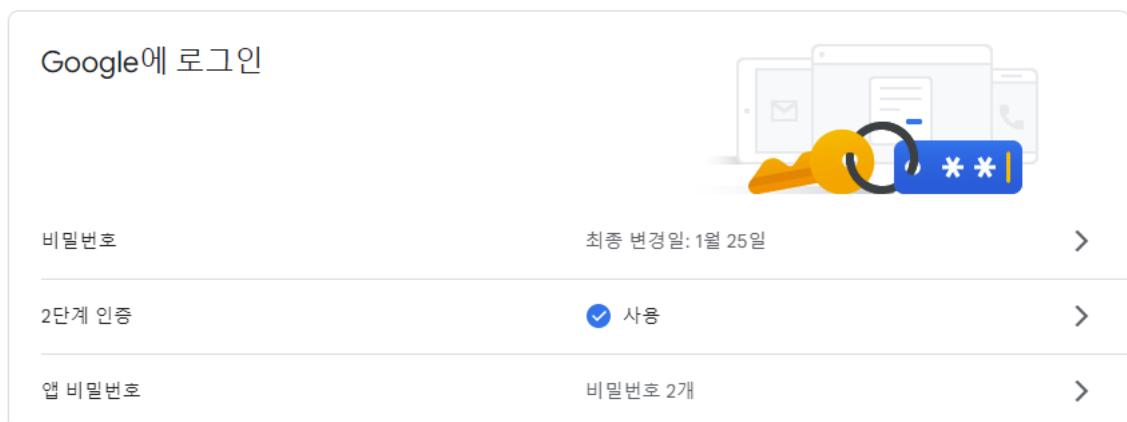
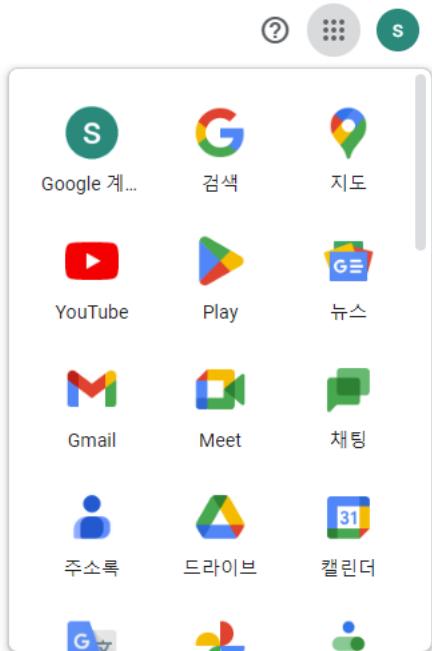
```

key:
  youtube: {API 키}

```

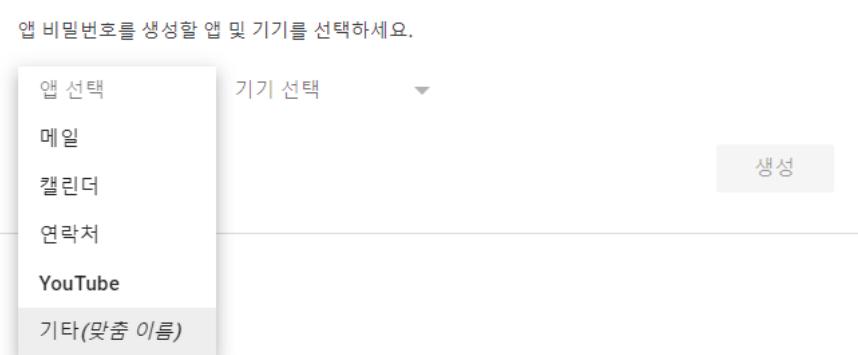
SMTP - 구글 메일 설정

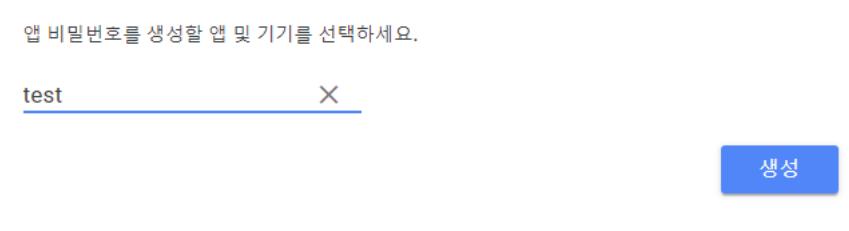
1. 구글 계정 생성
2. 프로필 클릭 → 계정 설정 → 2단계 인증 활성화



3. 2단계 인증 활성화 이후 앱 비밀번호 발급

- 앱 선택 : 기타(맞춤 이름)
- 앱 비밀번호를 생성할 이름 입력 후 생성





4. 앱 비밀번호 발급 후 앱 key 입력

생성된 앱 비밀번호

기기용 앱 비밀번호

Email
securesally@gmail.com

Password

사용 방법

설정하려는 애플리케이션 또는 기기의 Google 계정 설정으로 이동합니다. 비밀번호를 위에 표시된 16자리 비밀번호로 교체합니다.
일반적인 비밀번호와 마찬가지로 이 앱 비밀번호는 Google 계정에 대한 완전한 액세스 권한을 부여합니다. 비밀번호를 기억하지 않아도 되므로 적어 놓거나 다른 사용자와 공유하지 마세요.

확인

```
spring:  
  mail:  
    host: smtp.gmail.com  
    port: 587  
    username: {google 메일 아이디}  
    password: {google 계정 앱 key}
```

5. G메일 → 상단 설정 버튼 클릭 → 빠른 설정/모든 설정 보기 클릭 → 전달 및 POP/IMAP 클릭 → 이메일 보내기 설정

기본설정 라벨 받은편지함 계정 및 가져오기 끌티 및 차단된 주소 **전달 및 POP/IMAP** 부가기능 채팅 및 Meet 고급 오프라인 테마

전달:
자세히 알아보기

도움말: 필터를 만들면 매 일 중 일부만 전달할 수도 있습니다.

POP 다운로드:
자세히 알아보기

1. 상태: 모든 메일에 대해 POP을 사용 설정되어 있습니다.
 이미 다운로드 된 메일을 포함하여 모든 메일에 POP을 활성화 하기
 지금부터 수신되는 메일에만 POP을 사용하기
 POP 사용 안함

2. POP로 메시지를 여는 경우 [Gmail 사본을 받은편지함에 보관하기]

3. 이메일 클라이언트 구성 (예: Outlook, Eudora, Netscape Mail)
설정 방법

IMAP 액세스:
(IMAP를 사용하여 다른 클라이언트에서 Gmail에 액세스)
자세히 알아보기

상태: IMAP를 사용할 수 있습니다.
 IMAP 사용
 IMAP 사용 안함

IMAP에서 메일을 삭제된 것으로 표시하는 경우:
 자동 삭제 사용 - 서버를 즉시 업데이트(기본값)
 메일을 휴지통으로 이동
 메일을 즉시 완전삭제

폴더 크기 제한
 IMAP 폴더에서 메일 수를 제한하지 않습니다(기본값).
 이만큼의 메일만 포함하도록 IMAP 폴더를 제한합니다. 1,000

이메일 클라이언트 구성(예: Outlook, Thunderbird, iPhone)
설정 방법

[변경사항 저장] [취소]

S3 버킷

1. AWS 계정 생성

2. AWS 서비스 검색창에 S3 검색 및 이동

3. 버킷 클릭

4. 버킷 정보 입력

일반 구성

버킷 이름

shabit-test

버킷 이름은 전역에서 고유해야 하며 공백 또는 대문자를 포함할 수 없습니다. [버킷 이름 지정 규칙 참조](#)

AWS 리전

아시아 태평양(서울) ap-northeast-2



기존 버킷에서 설정 복사 - 선택 사항
다음 구성의 버킷 설정만 복사됩니다.

[버킷 선택](#)

객체 소유권 Info

다른 AWS 계정에서 이 버킷에 작성한 객체의 소유권 및 액세스 제어 목록(ACL)의 사용을 제어합니다. 객체 소유권은 객체에 대한 액세스를 지정 할 수 있는 사용자를 결정합니다.

ACL 비활성화됨(권장)

이 버킷의 모든 객체는 이 계정이 소유합니다. 이 버킷과 그 객체에 대한 액세스는 정책을 통해서만 지정됩니다.

ACL 활성화됨

이 버킷의 객체는 다른 AWS 계정에서 소유할 수 있습니다.
이 버킷 및 객체에 대한 액세스는 ACL을 사용하여 지정할 수 있습니다.

객체 소유권

버킷 소유자 적용



ACL 비활성화 관련 향후 권한 변경 사항

2023년 4월부터는 S3 콘솔을 사용하여 버킷을 생성할 때 ACL을 비활성화하기 위해 더 이상 s3:PutBucketOwnershipControls 권한이 필요하지 않습니다. [자세히 알아보기](#)

이 버킷의 퍼블릭 액세스 차단 설정

퍼블릭 액세스는 ACL(액세스 제어 목록), 버킷 정책, 액세스 지점 정책 또는 모두를 통해 버킷 및 객체에 부여됩니다. 이 버킷 및 해당 객체에 대한 퍼블릭 액세스가 차단되었는지 확인하려면 모든 퍼블릭 액세스 차단을 활성화합니다. 이 설정은 이 버킷 및 해당 액세스 지점에만 적용됩니다. AWS에서는 모든 퍼블릭 액세스 차단을 활성화하도록 권장하지만, 이 설정을 적용하기 전에 퍼블릭 액세스가 없어도 애플리케이션이 올바르게 작동하는지 확인합니다. 이 버킷 또는 내부 객체에 대한 어느 정도 수준의 퍼블릭 액세스가 필요한 경우 특정 스토리지 사용 사례에 맞게 아래 개별 설정을 사용자 지정할 수 있습니다. 자세히 알아보기 [\[\]](#)

모든 퍼블릭 액세스 차단

이 설정을 활성화하면 아래 4개의 설정을 모두 활성화한 것과 같습니다. 다음 설정 각각은 서로 독립적입니다.

새 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 새로 추가된 버킷 또는 객체에 적용되는 퍼블릭 액세스 권한을 차단하며, 기존 버킷 및 객체에 대한 새 퍼블릭 액세스 ACL 생성을 금지합니다. 이 설정은 ACL을 사용하여 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 권한을 변경하지 않습니다.

임의의 ACL(액세스 제어 목록)을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 모든 ACL을 무시합니다.

새 퍼블릭 버킷 또는 액세스 지점 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 새 버킷 및 액세스 지점 정책을 차단합니다. 이 설정은 S3 리소스에 대한 퍼블릭 액세스를 허용하는 기존 정책을 변경하지 않습니다.

임의의 퍼블릭 버킷 또는 액세스 지점 정책을 통해 부여된 버킷 및 객체에 대한 퍼블릭 및 교차 계정 액세스 차단

S3은 버킷 및 객체에 대한 퍼블릭 액세스를 부여하는 정책을 사용하는 버킷 또는 액세스 지점에 대한 퍼블릭 및 교차 계정 액세스를 무시합니다.



모든 퍼블릭 액세스 차단을 비활성화하면 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있습니다.

정적 웹 사이트 호스팅과 같은 구체적으로 확인된 사용 사례에서 퍼블릭 액세스가 필요한 경우가 아니면 모든 퍼블릭 액세스 차단을 활성화하는 것이 좋습니다.

- 현재 설정으로 인해 이 버킷과 그 안에 포함된 객체가 퍼블릭 상태가 될 수 있음을 알고 있습니다.

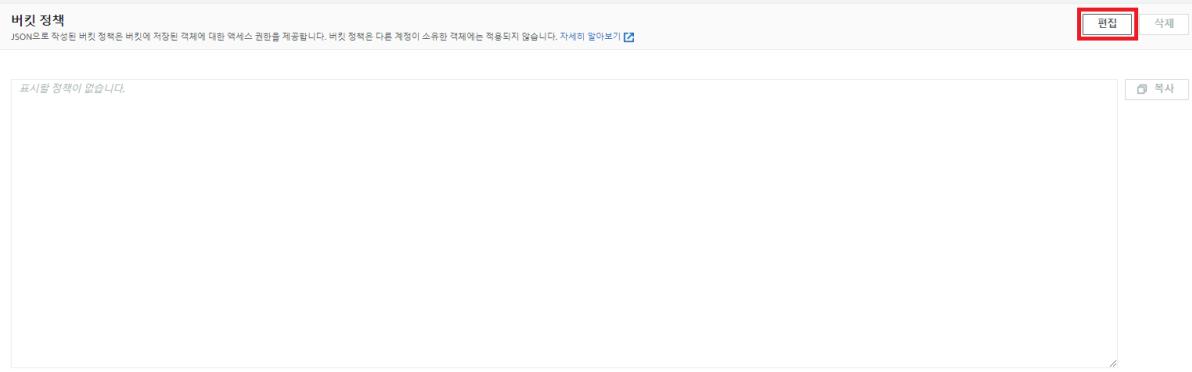


모든 퍼블릭 액세스 차단 설정 비활성화 관련 항후 권한 변경 사항

2023년 4월부터는 S3 콘솔을 사용하여 버킷을 생성할 때 퍼블릭 액세스 차단 설정을 비활성화하기 위해 s3:PutBucketPublicAccessBlock 권한이 있어야 합니다. 자세히 알아보기 [\[\]](#)

5. 버킷 설정 - 권한 설정

- 권한 클릭 → 버킷 정책 편집 클릭



- 권한 클릭 → 버킷 정책 편집 클릭 → 정책 생성기 클릭

버킷 정책
JSON으로 작성된 버킷 정책은 버킷에 저장된 객체에 대한 액세스 권한을 제공합니다. 버킷 정책은 다른 계정이 소유한 객체에는 적용되지 않습니다. 자세히 알아보기

버킷 ARN
`arn:aws:s3:::shabit-test`

정책

- Action에는 `GetObject`, `PutObject`, `DeleteObject` 3개를 체크하고 ARN에는 복사해둔 ARN값을 입력한다.
- ARN값을 입력하되 /* 값도 추가 해줘야한다. ARN 값이 `arn:aws:s3:::test`라 가정하면 `arn:aws:s3:::test/*` 라고 입력해주면 된다.

Step 1: Select Policy Type

A Policy is a container for permissions. The different types of policies you can create are an IAM Policy, an S3 Bucket Policy, an SQS Queue Policy.

Select Type of Policy S3 Bucket Policy

Step 2: Add Statement(s)

A statement is the formal description of a single permission. See a description of elements that you can use in statements.

Effect Allow Deny

Principal *
Use a comma to separate multiple values.

AWS Service Amazon S3 All Services ('*')

Actions 3 Action(s) Selected All Actions ('*')

Amazon Resource Name (ARN) arn:aws:s3:::shabit-test/*
ARN should follow the following format: `arn:aws:s3:::${BucketName}/${KeyName}`.
Use a comma to separate multiple values.

Add Conditions (Optional)

Add Statement

- 스크롤을 아래로 내려 Generate Policy 버튼을 클릭한다.
- 생성된 정책을 복사한뒤 정책 편집 부분에 붙여넣은 후 변경 사항 저장 버튼을 눌러 저장한다.

Policy JSON Document

Click below to edit. To save the policy, copy the text below to a text editor.
Changes made below will not be reflected in the policy generator tool.

```
{
  "Id": "Policy1676275942394",
  "Version": "2012-10-17",
  "Statement": [
    {
      "Sid": "Stmt1676275934557",
      "Action": [
        "s3:DeleteObject",
        "s3:GetObject",
        "s3:PutObject"
      ],
      "Effect": "Allow",
      "Resource": "arn:aws:s3:::shabit-test/*",
      "Principal": "*"
    }
  ]
}
```

Close

버킷 정책

JSON으로 작성된 버킷 정책은 버킷에 저장된 객체에 대한 액세스 권한을 제공합니다. 버킷 정책은 다른 계정이 소유한 객체에는 적용되지 않습니다. 자세히 알아보기 [\[↗\]](#)

버킷 ARN

arn:aws:s3:::shabit-test

정책

```

1  [
2   "Id": "Policy1676275982590",
3   "Version": "2012-10-17",
4   "Statement": [
5     {
6       "Sid": "Stmt1676275934557",
7       "Action": [
8         "s3:DeleteObject",
9         "s3:GetObject",
10        "s3:PutObject"
11       ],
12       "Effect": "Allow",
13       "Resource": "arn:aws:s3:::shabit-test/*",
14       "Principal": "*"
15     }
16   ]
17 ]
```

6. 사용자 등록

- AWS 서비스 검색창에 **IAM** 검색 → **사용자** 클릭

IAM > 사용자

사용자 (1) 정보

IAM 사용자는 계정에서 AWS와 상호 작용하는 데 사용되는 장기 자격 증명을 가진 자격 증명입니다.

7. 사용자 생성

8. 사용자 설정

- 보안 자격 증명 클릭 → 액세스 키 발급 클릭

요약

ARN arn:aws:iam::120626585696:user/shabit-user	콘솔 액세스 ⚠️ MFA 없이 활성화됨	액세스 키 1 활성화되지 않음
생성됨 February 13, 2023, 17:33 (UTC+09:00)	마지막 콘솔 로그인 ① 안 함	액세스 키 2 활성화되지 않음

권한 | 그룹 | 태그 | **보안 자격 증명** | 액세스 관리자

권한 정책 (1)
사용자에게 직접 연결된 정책을 통해 또는 그룹을 통해 권한을 정의합니다.

액세스 키 (0)
액세스 키를 사용하여 AWS CLI, AWS Tools for PowerShell, AWS SDK 또는 직접 AWS API 호출을 통해 AWS에 프로그래밍 방식 호출을 전송합니다. 한 번에 최대 두 개의 액세스 키(활성 또는 비활성)를 가질 수 있습니다. [Learn more](#)

액세스 키 만들기

액세스 키 없음
액세스 키와 같은 장기 자격 증명을 사용하지 않는 것이 모범 사례입니다. 대신 단기 자격 증명을 제공하는 도구를 사용하세요. [Learn more](#)

액세스 키 만들기

액세스 키 모범 사례 및 대안

보안 개선을 위해 액세스 키와 같은 장기 자격 증명을 사용하지 마세요. 다음과 같은 사용 사례와 대안을 고려하세요.

- Command Line Interface(CLI)**
AWS CLI를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.
- 로컬 코드**
로컬 개발 환경의 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.
- AWS 컴퓨팅 서비스에서 실행되는 애플리케이션**
Amazon EC2, Amazon ECS 또는 AWS Lambda와 같은 AWS 컴퓨팅 서비스에서 실행되는 애플리케이션 코드를 사용하여 AWS 계정에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.
- 서드 파티 서비스**
AWS 리소스를 모니터링 또는 관리하는 서드 파티 애플리케이션 또는 서비스에 액세스할 수 있도록 이 액세스 키를 사용할 것입니다.
- AWS 외부에서 실행되는 애플리케이션**
애플리케이션을 온프레미스 호스트에서 실행하거나 로컬 AWS 클라이언트 또는 서드 파티 AWS 플러그 인을 사용할 수 있도록 이 액세스 키를 사용할 것입니다.
- 기타**
귀하의 사용 사례가 여기에 나열되어 있지 않습니다.

설명 태그 설정 - 선택 사항

이 액세스 키에 대한 설명은 이 사용자에게 태그로 연결되고, 액세스 키와 함께 표시됩니다.

설명 태그 값

이 액세스 키의 용도와 사용 위치를 설명합니다. 좋은 설명은 나중에 이 액세스 키를 자신있게 교체하는 데 유용합니다.

최대 256자까지 가능합니다. 허용되는 문자는 문자, 숫자, UTF-8로 표현할 수 있는 공백 및 _ . : / = + - @입니다.

취소

이전

액세스 키 만들기

액세스 키 검색

액세스 키

분실하거나 잊어버린 비밀 액세스 키는 검색할 수 없습니다. 대신 새 액세스 키를 생성하고 이전 키를 비활성화합니다.

액세스 키

비밀 액세스 키

[]

[] 습기기

액세스 키 모범 사례

- 액세스 키를 일반 텍스트, 코드 리포지토리 또는 코드로 저장해사는 안됩니다.
- 더 이상 필요 없는 경우 액세스 키를 비활성화하거나 삭제합니다.
- 최소 권한을 활성화합니다.
- 액세스 키를 정기적으로 교체합니다.

액세스 키 관리에 대한 자세한 내용은 [AWS 액세스 키 관리 모범 사례](#)를 참조하세요.

.csv 파일 다운로드

완료

9. 발급받은 `access-key`, `secret-key` 입력

```
cloud:  
aws:  
  s3:  
    bucket: 버킷 이름  
region:  
  static: ap-northeast-2 #Asia Pacific -> seoul  
stack:  
  auto: false  
credentials:  
  access-key: S3 사용자 access-key  
  secret-key: S3 사용자 secret-key
```

ELK Stack + Kafka

로그가 수집되는 전체적인 흐름

Kafka 배포하기

Filebeat 배포하기

ELK 배포하기

로그 수집 확인하기

로그가 수집되는 전체적인 흐름

1. 스프링 서버에서 로그가 발생한다.
2. 발생한 로그는 파일 형식(.log)으로 지정된 디렉토리에 저장된다.
3. 해당 디렉토리를 관찰하고 있는 filebeat가 로그 파일이 생성된 것을 인식한다.
4. filebeat는 로그 파일을 logstash로 전송한다.
5. logstash는 전달받은 로그 파일을 파싱하여 elasticsearch로 전송한다.
6. kibana는 elasticsearch에 전달된 데이터들을 시각화하여 보여준다.

Kafka 배포하기

```
mkdir kafka  
cd kafka  
vi docker-compose.yml
```

```
version: '2'  
services:  
  zookeeper:  
    image: wurstmeister/zookeeper  
    ports:  
      - "2181:2181"  
  kafka:  
    image: wurstmeister/kafka  
    ports:  
      - "9092:9092"  
    environment:  
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://3.34.8.99:9092  
      KAFKA_LISTENERS: PLAINTEXT://:9092  
  
      KAFKA_CREATE_TOPICS: "log:1:1"  
      KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181  
    volumes:  
      - /var/run/docker.sock:/var/run/docker.sock  
    depends_on:  
      - zookeeper  
  kafka-ui:  
    image: provectuslabs/kafka-ui  
    container_name: kafka-ui  
    ports:  
      - "9090:8080"  
    restart: always  
    environment:  
      - KAFKA_CLUSTERS_0_NAME=local  
      - KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=kafka:9092  
      - KAFKA_CLUSTERS_0_ZOOKEEPER=zookeeper:2181
```

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

```
version: '2'  
services:  
  zookeeper:  
    hostname: zookeeper  
    container_name: zookeeper  
    image: wurstmeister/zookeeper  
    ports:  
      - "2181:2181"  
  
  kafka1:  
    hostname: kafka1  
    container_name: kafka1  
    image: wurstmeister/kafka  
    ports:  
      - "9092:9092"  
    environment:  
      KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://3.34.8.99:9092  
      KAFKA_LISTENERS: PLAINTEXT://:9092
```

```

KAFKA_BROKER_ID: 1
KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
BOOTSTRAP_SERVERS: kafka1:9092, kafka2:9093, kafka3:9094
KAFKA_CREATE_TOPICS: "user-log:1:1,user-req:5:1,user-res:1:1"
volumes:
- /var/run/docker.sock:/var/run/docker.sock
depends_on:
- zookeeper
kafka2:
hostname: kafka2
container_name: kafka
image: wurstmeister/kafka
ports:
- "9093:9092"
environment:
KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://3.34.8.99:9093
KAFKA_LISTENERS: PLAINTEXT://:9092
KAFKA_BROKER_ID: 2
KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
BOOTSTRAP_SERVERS: kafka1:9092, kafka2:9093, kafka3:9094
KAFKA_CREATE_TOPICS: "payment-log:1:1,payment-req:5:1,payment-res:1:1"
volumes:
- /var/run/docker.sock:/var/run/docker.sock
depends_on:
- zookeeper
kafka3:
hostname: kafka3
container_name: kafka
image: wurstmeister/kafka
ports:
- "9094:9092"
environment:
KAFKA_ADVERTISED_LISTENERS: PLAINTEXT://3.34.8.99:9094
KAFKA_LISTENERS: PLAINTEXT://:9092
KAFKA_BROKER_ID: 3
KAFKA_ZOOKEEPER_CONNECT: zookeeper:2181
BOOTSTRAP_SERVERS: kafka1:9092, kafka2:9093, kafka3:9094
KAFKA_CREATE_TOPICS: "concert-log:1:1,concert-req:5:1,concert-res:1:1"
volumes:
- /var/run/docker.sock:/var/run/docker.sock
depends_on:
- zookeeper
kafka-ui:
image: provectuslabs/kafka-ui
container_name: kafka-ui
ports:
- "8080:8080"
restart: always
environment:
- KAFKA_CLUSTERS_0_NAME=local
- KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS=kafka1:9092,kafka2:9092,kafka3:9092
- KAFKA_CLUSTERS_0_ZOOKEEPER=zookeeper:2181

```

Filebeat 배포하기

배포 환경

- AWS ec2 (3.34.8.99)
- `~/config` 디렉토리에서 진행

Spring 서버 설정하기

1. 라이브러리 추가

log message를 ELK가 읽을 수 있는 json형태의 로그로 변경을 위해 `logstash-logback-encoder`를 추가해주어야 한다.

```
// https://mvnrepository.com/artifact/net.logstash.logback/logstash-logback-encoder
implementation group: 'net.logstash.logback', name: 'logstash-logback-encoder', version: '6.3'
```

2. logback 설정 (application.yml)

```
logging:
  level:
    com:
      allback:
        cygiconcert: info
  config: classpath:logback.xml
```

logback.xml 파일의 위치를 알려준다.

3. logback.xml

```
<?xml version="1.0" encoding="UTF-8"?>
<configuration>
  <property name="CONSOLE_LOG_PATTERN"
    value="%d{yyyy-MM-dd HH:mm:ss.SSS} %highlight(%-5level) %magenta(%-4relative) --- [ %thread{10} ] %cyan(%logger{20}) : %>
  <!--프로젝트 최상단을 기준으로 아래 경로에 로그 파일(.log)이 저장된다.-->
  <property name="LOG_PATH" value="/var/log/cygi"/>
  <!--저장되는 로그 파일의 이름을 지정한다.-->
  <!--저장되는 로그 파일의 이름을 지정한다.-->
  <property name="FILE_NAME" value="cygi-concert-logs"/>

  <appender name="CONSOLE" class="ch.qos.logback.core.ConsoleAppender">
    <encoder class="ch.qos.logback.classic.encoder.PatternLayoutEncoder">
      <pattern>${CONSOLE_LOG_PATTERN}</pattern>
    </encoder>
  </appender>

  <appender name="FILE" class="ch.qos.logback.core.rolling.RollingFileAppender">
    <file>${LOG_PATH}/${FILE_NAME}-json.log</file>
    <encoder class="net.logstash.logback.encoder.LogstashEncoder"/>
    <rollingPolicy class="ch.qos.logback.core.rolling.TimeBasedRollingPolicy">
      <fileNamePattern>${LOG_PATH}/${FILE_NAME}_%d{yyyy-MM-dd}.%i.log</fileNamePattern>
      <maxHistory>90</maxHistory>
      <timeBasedFileNamingAndTriggeringPolicy>
        <class>ch.qos.logback.core.rolling.SizeAndTimeBasedFNATP</class>
        <maxFileSize>10MB</maxFileSize>
      </timeBasedFileNamingAndTriggeringPolicy>
    </rollingPolicy>
  </appender>

  <root level = "INFO">
    <appender-ref ref="FILE"/>
    <appender-ref ref="CONSOLE"/>
  </root>
</configuration>
```

로그 파일이 어느 디렉토리에 저장될 것인지, 어떤 파일명으로 저장될 것인지 등을 설정한다.

4. Spring 서버 배포하기

Spring 서버를 배포하는 다른 잡다한 과정은 여기서는 생략한다.

단, Spring 서버를 도커 컨테이너에 올릴 때 volume 옵션을 꼭 추가해야 한다.

-v [volume 이름]:[log 파일이 저장되는 디렉토리 경로]

- volume 이름은 적당히 자신이 알아볼 수 있는 이름으로 정하면 된다.

- log 파일이 저장되는 디렉토리 경로는 아까 'logback.xml' 파일에서 지정한 log 저장 경로를 기입하면 된다.

예 : `docker run -v log-concert:/var/log/cygi --name back-concert -d -p 8002:8090 back-concert:${env.BUILD_NUMBER}`

Filebeat 배포하기

filebeat를 설치할 수 있는 경우의 수는 다양하다.

1. Spring 도커 컨테이너 안에 filebeat를 직접 설치하기
2. 호스트에 filebeat를 직접 설치하기
3. filebeat를 따로 도커 컨테이너로 띄우기

여기서는 3번 방법으로 filebeat를 설치한다.

filebeat 폴더 생성

```
mkdir filebeat
cd filebeat
```

Spring 서버가 총 3개(concert, user, payment)가 실행되고 있기 때문에, 각 Spring 서버 별로 filebeat 컨테이너를 1개씩 띄울 예정이다. 따라서 back-concert, back-user, back-payment 디렉토리를 각각 만들고, Dockerfile과 filebeat.yml 파일도 각각 만들어서 컨테이너를 각각 띄울 것이다.

```
mkdir back-concert
mkdir back-user
mkdir back-payment
```

여기서는 concert 버전으로 filebeat를 배포하는 것만 설명한다.

아래부터는 user와 payment 버전으로 2번 더 따라하면 된다.

```
cd back-concert
```

Dockerfile

```
vi Dockerfile
```

```
FROM docker.elastic.co/beats/filebeat:7.12.0
COPY filebeat.yml /usr/share/filebeat/filebeat.yml
USER root
RUN mkdir /var/log/cygi
RUN chmod go-w /usr/share/filebeat/filebeat.yml
```

filebeat.yml

```
vi filebeat.yml
```

```

filebeat.inputs:
  - type: log
    enabled: true
    paths:
      - "/var/log/cygi/*.log"
    tags: ["concert-log"]
    fields: {log_type: concert, index_type: concert-log}
  - type: log
    paths:
      - "/var/log/cygi/access/*.log"
    tags: ["concert-access"]
    fields: {log_type: concert, index_type: concert-access}
    exclude_lines: [".*ELB-HealthChecker.*"]
  - type: log
    paths:
      - "/var/log/cygi/error/*.log"
    tags: ["concert-error"]
    fields: {log_type: concert, index_type: concert-error}

setup.dashboards.enabled: false

#output.logstash:
#hosts: ["k8a806.p.ssafy.io:8044"]
# hosts: ["logstash:5000"] # docker-elk 내부에서 띄울 경우

output.kafka:
  hosts: ["kafka:9092"]
  topic: "log"
  partition.round_robin:
    reachable_only: false
  required_acks: 1

```

filebeat 실행하기

filebeat.yml 파일을 수정했으면 도커 이미지 삭제하고 다시 빌드해야한다.

```
docker rmi -f filebeat-concert
```

```
docker build --no-cache --tag filebeat-concert .
```

```
docker run -d -p 5042:5044 -v log-concert:/var/log/cygi --name filebeat-concert --network kafka_default filebeat-concert
```

filebeat를 도커 컨테이너로 띄울 때 volume 옵션에 주의해야한다.

```
-v [volume 이름]:[log 파일을 관찰할 디렉토리 경로]
```

- volume 이름은 아까 Spring 서버를 컨테이너로 띄울 때 썼던 volume 이름과 동일하게 써줘야한다.
- log 파일을 관찰할 디렉토리 경로는 filebeat.yml 파일에서 써놓은 path경로와 일치시킨다.

ELK 배포하기

배포 환경

- AWS ec2 (k8a806.p.ssafy.io)
- `~/config` 디렉토리에서 진행

elk 설정 파일을 저장할 디렉토리를 생성하고 이동한다.

```
mkdir elk  
cd elk
```

docker compose를 이용하여 한꺼번에 배포할 것이므로, docker compose 설정 파일을 구성한다.

```
vi docker-compose.yml
```

```
version: '2'  
  
services:  
  elasticsearch:  
    image: docker.elastic.co/elasticsearch/elasticsearch:7.12.0  
    container_name: elasticsearch  
    volumes:  
      - ./elasticsearch/config/elasticsearch.yml:/usr/share/elasticsearch/config/elasticsearch.yml:ro  
      - /etc/localtime:/etc/localtime:ro  
    environment:  
      - discovery.type=single-node  
    ports:  
      - "9200:9200"  
      - "9300:9300"  
    networks:  
      - elk  
  logstash:  
    image: docker.elastic.co/logstash/logstash:7.12.0  
    container_name: logstash  
    volumes:  
      - ./logstash/config/logstash.yml:/usr/share/logstash/config/logstash.yml  
      - ./logstash/pipeline:/usr/share/logstash/pipeline  
      - /etc/localtime:/etc/localtime:ro  
    ports:  
      - "8044:5044"  
      - "8000:5000"  
    networks:  
      - elk  
    extra_hosts:  
      - "kafka:3.34.8.99"  
  kibana:  
    image: docker.elastic.co/kibana/kibana:7.12.0  
    container_name: Kibana  
    ports:  
      - "8601:5601"  
    environment:  
      - ELASTICSEARCH_URL=http://elasticsearch:9200  
      - ELASTICSEARCH_HOSTS=http://elasticsearch:9200  
    volumes:  
      - ./kibana/config/kibana.yml:/usr/share/kibana/config/kibana.yml:ro  
      - /etc/localtime:/etc/localtime:ro  
    networks:  
      - elk  
networks:  
  elk:  
    driver: bridge
```

logstash 관련 설정

```
mkdir logstash  
mkdir logstash/config  
mkdir logstash/pipeline  
vi logstash/config/logstash.yml
```

```
http.host: "0.0.0.0"  
xpack.monitoring.enabled: true  
  
# 추측  
# 이게 없으면 디폴트로 logstash는 "localhost:9200"과 연결을 시도하는 듯 하다.  
# logstash와 elasticsearch는 같은 컨테이너에서 작동하지 않으므로 localhost로 통신할 수 없다.  
xpack.monitoring.elasticsearch.hosts: ["http://elasticsearch:9200"]
```

```
vi logstash/pipeline/01-input.conf
```

```

input {
    # 로그를 수신할 위치 (filebeat 아님 kafka)
    #beats {
        #port => 5044
        #host => "0.0.0.0"
    //}
    kafka {
        bootstrap_servers => "3.34.8.99:9092"
        topics => ["log"]
    }
}
#filter {
#    json {
#        source => "message"
#    }
#    json {
#        source => "level"
#    }
#}
output {
    # 처리한 로그를 Elastic 서버로 전송
    elasticsearch {
        # TODO 각자의 서버에 맞게 IP 변경
        hosts => ["k8A806.p.ssafy.io:9200"]
        index => "logstash-%{+YYYY.MM.dd}"
    }
}

```

elasticsearch 관련 설정

```
mkdir elasticsearch
```

```
mkdir elasticsearch/config
```

```
vi elasticsearch/config/elasticsearch.yml
```

```

---
## Default Elasticsearch configuration from Elasticsearch base image.
cluster.name: "docker-cluster"
network.host: 0.0.0.0

## Use single node discovery in order to disable production mode and avoid bootstrap checks
discovery.type: single-node

# 보안 관련 설정 해제
xpack.ml.enabled: false

```

kibana 관련 설정

```
mkdir kibana
```

```
mkdir kibana/config
```

```
vi kibana/config/kibana.yml
```

```

---
## Default Kibana configuration from Kibana base image.
server.name: kibana
server.host: "0.0.0.0"
elasticsearch.hosts: [ "http://k8A806.p.ssafy.io:9300" ]

```

ELK 실행하기

```
sudo docker-compose up -d
```

로그 수집 확인하기

Spring 컨테이너 내부에 접속해서 log 파일이 저장하는 경로로 이동한 후, 아무 log 파일이나 생성해본다.

elasticsearch에서 확인하기

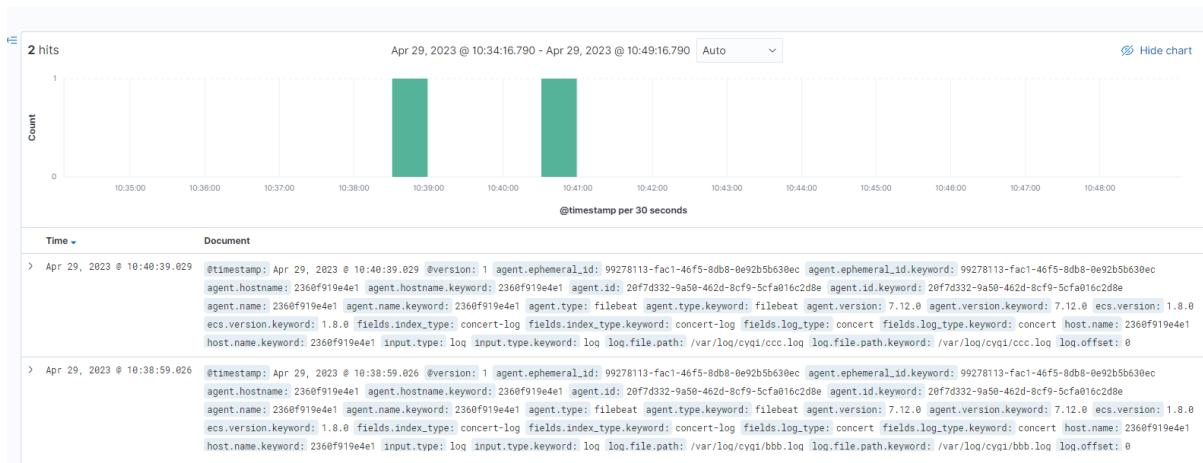
http://k8a806.p.ssafy.io:9200/_cat/indices

위 링크에 접속했을 때 logstash에서 설정한 index 데이터가 들어있다면 `spring 서버` → `filebeat` → `logstash` → `elasticsearch` 까지 무사히 데이터가 전달된 것이다.

kibana에서 확인하기

<http://k8a806.p.ssafy.io:5601>

kibana에서 데이터가 들어온 것을 확인하려면 logstash에서 설정해준 index를 kibana에서 생성해주어야한다.



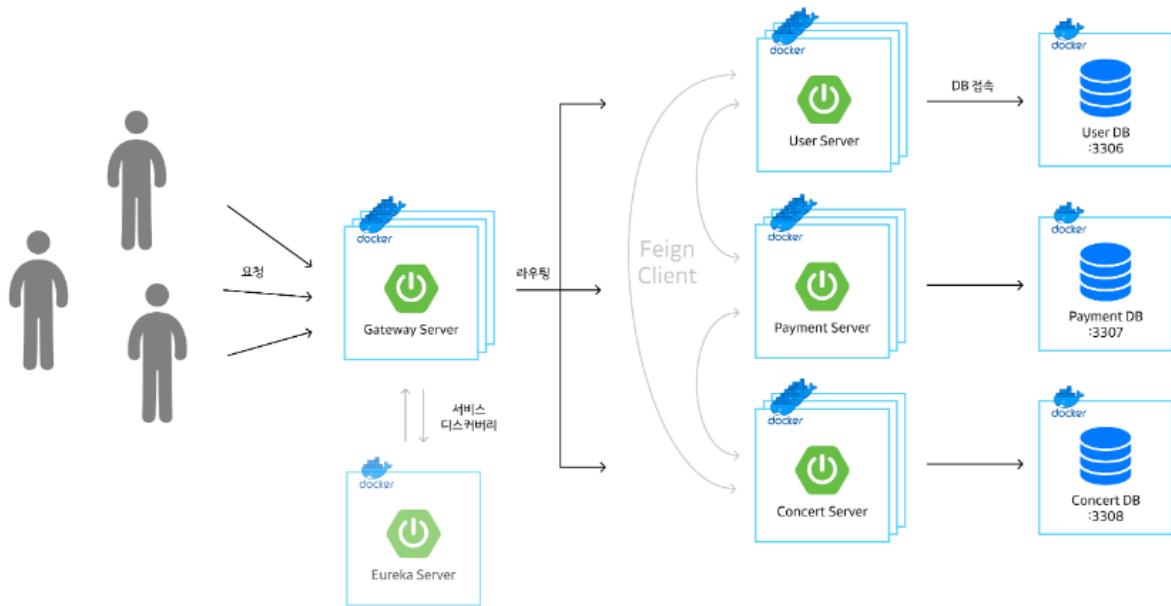
MSA 분산 서비스 구축

spring 서버 및 DB를 기능별로 분리

MSA란 [MicroService Architecture](#)의 약자로, 기존의 Monolithic Architecture의 한계를 벗어나 애플리케이션을 느슨하게 결합된 서비스의 모임으로 구조화하는 서비스 지향 아키텍처(SOA) 스타일의 일종인 소프트웨어 개발 기법입니다.

기능을 크게 4가지로 분류하고, 서버와 데이터베이스를 각 기능에 맞게 분류하여 구현하였습니다.

기능	Server	DB
사용자	User Server	User DB
공연	Concert Server	Concert DB
결제	Payment Server	Payment DB
관리자	Admin Server	



저희는 MSA 를 통해 다음과 같은 장점을 가질 수 있었습니다.

1. 배포

- 서비스별 개별 배포가 가능합니다.
- 특정 서비스의 요구사항만을 반영하여, 빠르게 배포 가능합니다.

2. 확장

- 특정 서비스에 대한 확장성(scale-out)이 유리합니다.
- 클라우드 기반 서비스 사용에 적합합니다.

3. 장애

- 일부 장애가 전체 서비스로 확장될 가능성은 적습니다.
- 부분적으로 발생하는 장애에 대한 격리가 수월합니다.

4. 그 외

- 새로운 기술을 적용하기 유연합니다.
- 각각의 서비스에 대한 구조 파악 및 분석이 모듈리식 구조에 비해 쉽습니다.

또한, MSA 를 구현하기 위해 각 기능과 역할별로 구분하여 브랜치를 구성하였습니다.

Branch Name Convention : (deploy/dev) - (back/front) - (service)

s08-final > Can You Get It > Repository > Branches	
	All
dev-back-payment	protected
dev-back-admin	protected
dev-back-user	protected
dev-back-concert	protected
dev-back-gateway	protected
dev-back-batch	protected
dev-back-discovery	protected
deploy-back-gateway	protected
deploy-back-concert	protected
deploy-back-payment	protected
deploy-back-discovery	protected
deploy-back-admin	merged
deploy-back-user	merged

Kubernetes 적용

k8s 수동 설치

모든 노드에서 다음 명령을 실행한다.

```
sudo swapoff -a # 현재 시스템에 적용(리부팅하면 재설정 필요)
sudo sed -i '/ swap / s/^(\.*\)$/#\1/g' /etc/fstab # 리부팅 필수
```

컨테이너 런타임 구성

모든 노드에 다음 명령으로 구성을 수행한다.

<https://www.itzgeek.com/how-tos/linux/ubuntu-how-tos/install-containerd-on-ubuntu-22-04.html>

```
# Using Docker Repository
sudo apt update
sudo apt install -y ca-certificates curl gnupg lsb-release
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /usr/share/keyrings/docker-archive-keyring.gpg
echo "deb [arch=$(dpkg --print-architecture) signed-by=/usr/share/keyrings/docker-archive-keyring.gpg] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list &> /dev/null

# containerd 설치
sudo apt update
sudo apt install -y containerd.io
# sudo systemctl status containerd # Ctrl + C를 눌러서 나간다.

# Containerd configuration for Kubernetes
cat <<EOF | sudo tee -a /etc/containerd/config.toml
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc]
[plugins."io.containerd.grpc.v1.cri".containerd.runtimes.runc.options]
SystemdCgroup = true
EOF

sudo sed -i 's/^disabled_plugins \=/\#disabled_plugins \=/g' /etc/containerd/config.toml
sudo systemctl restart containerd

# 소켓이 있는지 확인한다.
ls /var/run/containerd/containerd.sock
```

kubeadm, kubelet 및 kubectl 설치

모든 머신에 다음 패키지들을 설치한다.

- `kubeadm`: 클러스터를 부트스트랩하는 명령이다. 클러스터를 초기화하고 관리하는 기능을 갖는다.
- `kubelet`: 클러스터의 모든 머신에서 실행되는 파드와 컨테이너 시작과 같은 작업을 수행하는 컴포넌트이다. 데몬으로 동작하며 컨테이너를 관리한다.
- `kubectl`: 클러스터와 통신하기 위한 커맨드 라인 유ти리티이다. 클라이언트 전용 프로그램이다.

모든 노드에서 다음 명령을 실행한다.

```
cat <<EOF > kube_install.sh
# 1. apt 패키지 색인을 업데이트하고, 쿠버네티스 apt 리포지터리를 사용하는 데 필요한 패키지를 설치한다.
sudo apt-get update
sudo apt-get install -y apt-transport-https ca-certificates curl

# 2. 구글 클라우드의 공개 사이닝 키를 다운로드 한다.
sudo curl -fsSLo /usr/share/keyrings/kubernetes-archive-keyring.gpg https://packages.cloud.google.com/apt/doc/apt-key.gpg

# 3. 쿠버네티스 apt 리포지터리를 추가한다.
echo "deb [signed-by=/usr/share/keyrings/kubernetes-archive-keyring.gpg] https://apt.kubernetes.io/ kubernetes-xenial main" | sudo tee /etc/apt/sources.list.d/kubernetes.list

# 4. apt 패키지 색인을 업데이트하고, kubelet, kubeadm, kubectl을 설치하고 해당 버전을 고정한다.
sudo apt-get update
sudo apt-get install -y kubelet kubeadm kubectl
sudo apt-mark hold kubelet kubeadm kubectl
EOF

sudo bash kube_install.sh
```

`kubeadm` 버전을 확인한다.

```
kubeadm version
```

넷필터 브릿지 설정

모든 노드에 다음 명령으로 넷필터 브릿지 설정

```
sudo -i
modprobe br_netfilter
echo 1 > /proc/sys/net/ipv4/ip_forward
echo 1 > /proc/sys/net/bridge/bridge-nf-call-iptables
exit
```

클러스터 구성

지금까지는 모든 노드에 실행되는 작업이 동일했다. 하지만 여기부터는 각 노드에서 작업하는 내용이 다르기 때문에 주의가 필요하다.

마스터 노드 초기화

`마스터 노드`에서 init 작업을 시작한다.

```
sudo kubeadm init
```

3~4분 정도 기다리면 다음과 같이 init에 성공한다. 성공하면 다음과 같이 가이드가 나타난다.

```
Your Kubernetes control-plane has initialized successfully!

# 1) 유저 설정
To start using your cluster, you need to run the following as a regular user:

mkdir -p $HOME/.kube
```

```

sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

Alternatively, if you are the root user, you can run:

export KUBECONFIG=/etc/kubernetes/admin.conf

# 2) 파드 네트워크 설정
You should now deploy a pod network to the cluster.
Run "kubectl apply -f [podnetwork].yaml" with one of the options listed at:
https://kubernetes.io/docs/concepts/cluster-administration/addons/

# 3) 워커 노드 조인 방법
Then you can join any number of worker nodes by running the following on each as root:

kubeadm join 10.142.0.3:6443 --token xcs79e.vnernooln6yyimtv \
    --discovery-token-ca-cert-hash sha256:c9f8642746515eadc28e72c687eface2fa64da93ddca5a30b4ccf931dbcce839

```

유저 설정

다음 명령을 실행하면 `.kube/config` 파일이 설정되면서 사용이 가능해진다. 이 실습은 `마스터 노드`에서만 진행한다.

```

mkdir -p $HOME/.kube
sudo cp -i /etc/kubernetes/admin.conf $HOME/.kube/config
sudo chown $(id -u):$(id -g) $HOME/.kube/config

```

인증 정보를 사용해 노드 목록을 확인한다.

```

$ kubectl get nodes
NAME      STATUS      ROLES      AGE      VERSION
master-1  NotReady   control-plane   5m      v1.26.0

```

워커 노드 조인하기

본인의 콘솔에 출력된 토큰과 hash 값을 사용해 조인을 수행해야 한다. 다음 명령은 `워커 노드`에서 실습을 진행한다.

```

sudo kubeadm join 10.142.0.3:6443 --token xcs79e.vnernooln6yyimtv \
    --discovery-token-ca-cert-hash sha256:c9f8642746515eadc28e72c687eface2fa64da93ddca5a30b4ccf931dbcce839

```

`마스터 노드`에서 노드 목록을 조회한다.

```

$ kubectl get nodes
NAME      STATUS      ROLES      AGE      VERSION
master-1  NotReady   control-plane   7m55s   v1.26.0
worker-1  NotReady   <none>     14s     v1.26.0
worker-2  NotReady   <none>     23s     v1.26.0

```



init이나 join을 잘못 수행한 경우

`sudo kubeadm reset`을 사용해 초기 설정으로 돌아갈 수 있다.



token 재발급 받는 방법 (마스터 노드에서 실습)

- 토큰 리스트 확인하기: `sudo kubeadm token list`
- 토큰 재발급하기: `sudo kubeadm token create --print-join-command`

파드 네트워크 배포

[마스터 노드](#)에서 다음 명령을 실행하면 앞서 구성한 유저 설정을 통해 클러스터에 cilium을 설치한다.

<https://kubernetes.io/docs/tasks/administer-cluster/network-policy-provider/cilium-network-policy/>

```
curl -LO https://github.com/cilium/cilium-cl/releases/latest/download/cilium-linux-amd64.tar.gz  
sudo tar xzvfC cilium-linux-amd64.tar.gz /usr/local/bin  
rm cilium-linux-amd64.tar.gz  
cilium install
```

설치가 잘 되었는지 확인한다.

노드의 상태가 레디로 변경되었는지 확인한다.

```
$ kubectl get nodes
NAME        STATUS    ROLES      AGE       VERSION
master-1   Ready     control-plane   31m      v1.26.0
worker-1   Ready     <none>      24m      v1.26.0
worker-2   Ready     <none>      24m      v1.26.0
```

GKE 클러스터 생성하기

console.cloud.google.com 으로 접속한다.

The screenshot shows the Google Cloud Platform interface. The top navigation bar includes 'Google Cloud' and a dropdown for 'gasbugs-kubernetes-20221219'. A search bar contains the placeholder '리소스, 문서, 제품 등을 검색하세요.' To the right are icons for notifications (6), help, and more.

The main menu on the left has 'Compute Engine' selected, indicated by a red underline. Below it are sections for '가상 머신' (Virtual Machines), 'VM 인스턴스' (VM Instances), '인스턴스 템플릿' (Instance Templates), '단독 태넌트 노드' (Single Tenant Node), '머신 이미지' (Machine Images), and 'TPU'.

The 'VM 인스턴스' section is active, showing a table of instances. One instance, 'instance-1', is selected, highlighted with a red box around its checkboxes in the first two columns. The table columns are: 선택 (Selected), 상태 (Status), 이름 (Name), 지역 (Region), 권장사항 (Recommendation), 다음에서 사용 중 (Used in next), 내부 IP (Internal IP), 외부 IP (External IP), and 연결 (Connect). The 'instance-1' row shows 'running', 'instance-1', 'us-central1-a', '10.128.0.2 (nic0)', and 'SSH'.

On the right side of the table, there are several buttons: 정지 (Stop), 재설정 (Reset), and three others (삭제 - Delete, 일정 만들기 - Create scheduled delete, and ?). The '삭제' button is also highlighted with a red box.

세 번의 클릭을 수행하면 구글의 기능을 사용해 클러스터를 구성할 수 있다. [만들기](#) 를 클릭한다.

The screenshot shows the Google Cloud Platform interface for the Kubernetes Engine. The left sidebar has 'Clusters' selected. The main area is titled 'Kubernetes Clusters' with a sub-section 'Kubernetes Engine'. A prominent red box surrounds the 'Create Cluster' button at the top right of the main content area.

스탠다드 모드로 클러스터를 **구성** 한다.

상세 설정은 그대로 두고 **만들기** 를 눌러 클러스터를 생성한다.

This screenshot shows the 'Cluster Configuration' step of the 'Create Cluster' wizard. It includes fields for 'Cluster Name' (set to 'cluster-1'), 'Region' (set to 'us-central1-c'), and 'Node Pool' (selected 'default-pool'). The 'Create' button at the bottom is highlighted with a red box.

클러스터가 생성되는데 5분 정도 소요된다.

클라우드 셀 활화

클라우드 셀은 GCP가 무료로 제공하는 VM이다. 상단의 콘솔 아이콘을 눌러서 활성화할 수 있다.

This screenshot shows the Google Cloud Platform dashboard with the 'Cloud Shell' terminal open. The terminal window is highlighted with a red box. It displays a welcome message from Cloud Shell, including the project name 'gasbugs-kubernetes-20221219' and instructions to type 'help' to get started.

새로 구성한 VM을 사용해 클러스터에 접근하려면 옆에 ...을 눌러서 연결 버튼을 누른다.

Kubernetes 클러스터 + 만들기 + 배포 ⚡ 새로고침

개요 관측 가능성 비용 최적화

필터 속성 이름 또는 값 입력

상태	이름 ↑	위치	노드 수	총 vCPU	총 메모리	알림	라벨
	cluster-1	us-central1-c	3	0	0GB	-	

연결

클라우드 셸에서 실행 버튼을 누르면 클라우드 셸에 이 명령이 자동 복사된다.

복제된 명령을 엔터를 눌러서 실행한다.

```
Welcome to Cloud Shell! Type "help" to get started.  
Your Cloud Platform project in this session is set to gasbugs-kubernetes-20221219.  
Use "gcloud config set project [PROJECT_ID]" to change to a different project.  
gasbugs2lc@cloudshell:~ (gasbugs-kubernetes-20221219)$ gcloud container clusters get-credentials cluster-1 --zone us-central1-c --project gasbugs-kubernetes-20221219   
Fetching cluster endpoint and auth data.  
WARNING: cluster cluster-1 is not RUNNING. The kubernetes API may or may not be available. Check the cluster status for more information.  
kubeconfig entry generated for cluster-1.  
gasbugs2lc@cloudshell:~ (gasbugs-kubernetes-20221219)$ 
```

클러스터가 정상적으로 실행되고 있다면 다음 명령으로 서버의 리스트를 조회할 수 있다.

kubectl get nodes

Google Cloud 리소스, 문서, 제품 등을 검색하세요. 검색

Kubernetes Engine Kubernetes 클러스터 + 만들기 + 배포 ⚡ 새로고침

클러스터 자주 묻는 질문 Marketplace 출시 노트

개요 관측 가능성 비용 최적화

필터 속성 이름 또는 값 입력

상태	이름 ↑	위치	노드 수	총 vCPU	총 메모리	알림	라벨
	cluster-1	us-central1-c	3	6	12GB	-	

CLOUD SHELL 터미널 (gasbugs-kubernetes-20221219) +

```
gasbugs2lc@cloudshell:~ (gasbugs-kubernetes-20221219)$ kubectl get nodes  
NAME STATUS ROLES AGE VERSION  
gke-cluster-1-default-pool-30c0fe5a-6ppg Ready <none> 100s v1.24.7-gke.900  
gke-cluster-1-default-pool-30c0fe5a-klzl Ready <none> 100s v1.24.7-gke.900  
gke-cluster-1-default-pool-30c0fe5a-v912 Ready <none> 97s v1.24.7-gke.900
```

helm chart

- 디렉토리 구조

```
├── README.md  
└── admin  
    └── Chart.yaml  
    └── templates  
        └── deployment.yaml
```

```

|   |   └── service.yaml
|   └── values.yaml
└── concert
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   └── service.yaml
    └── values.yaml
└── gateway
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   └── ingress.yaml
    └── service.yaml
    └── values.yaml
└── payment
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   └── service.yaml
    └── values.yaml
└── user
    ├── Chart.yaml
    ├── templates
    │   ├── deployment.yaml
    │   └── service.yaml
    └── values.yaml

```

- 예시) 다음과 같은 양식으로 작성

```

deployment.yaml

apiVersion: apps/v1
kind: Deployment
metadata:
  name: admin-deployment
spec:
  replicas: {{ .Values.admin.replicas }}
  selector:
    matchLabels:
      app: admin
  template:
    metadata:
      labels:
        app: admin
    spec:
      containers:
        - name: admin
          image: {{ .Values.admin.image.repository }}:{{ .Values.admin.image.tag }}
          ports:
            - containerPort: {{ (index .Values.admin.ports 0).port }}

```

```

service.yaml

apiVersion: v1
kind: Service
metadata:
  name: admin-service
spec:
  selector:
    app: admin
  ports:
    - protocol: TCP
      port: {{ (index .Values.admin.ports 0).port }}
      targetPort: {{ (index .Values.admin.ports 0).targetPort }}
  type: ClusterIP

```

```

Chart.yaml

apiVersion: v2
name: admin-chart
version: 1.0.0
appVersion: 1.0.0
description: SSAFY A806 Admin Helm Chart
maintainers:
  - name: Wonjoon Seong
    email: cadqe13@gmail.com

```

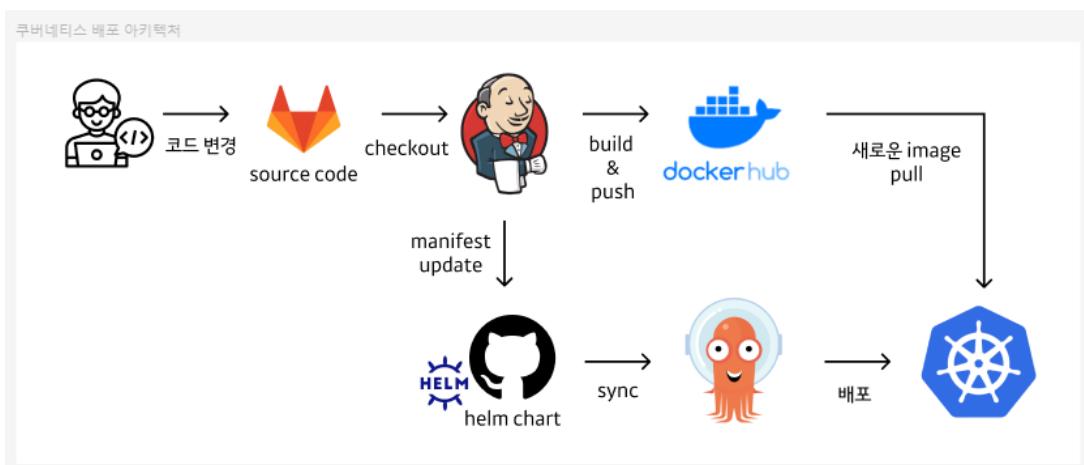
```

values.yaml

admin:
  enabled: true
  replicas: 3
  ports:
    - name: admin
      port: 80
      targetPort: 8090
  image:
    repository: wlwlsus/back-admin
    tag: 12

```

쿠버네티스 배포



Argo 설치

```

kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml

```

- argo를 쿠버네티스에 배포하자. 쿠버네티스에 argo를 배포하면 argo는 설치된 kube-api와 통신하며 클러스터를 관리한다. yaml 파일 내에 필요한 권한이 설정되어 있다.

설치 확인

```

$ kubectl get svc,pod -n argocd
NAME                           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)          AGE
service/argocd-dex-server     ClusterIP 10.8.10.139 <none>        5556/TCP,5557/TCP,5558/TCP 35m
service/argocd-metrics        ClusterIP 10.8.7.230  <none>        8082/TCP           35m
service/argocd-redis          ClusterIP 10.8.12.102 <none>        6379/TCP           35m
service/argocd-repo-server    ClusterIP 10.8.2.139  <none>        8081/TCP,8084/TCP 34m
service/argocd-server         ClusterIP 10.8.9.12   <none>        80/TCP,443/TCP   34m
service/argocd-server-metrics ClusterIP 10.8.13.88 <none>        8083/TCP           34m

NAME                           READY   STATUS    RESTARTS   AGE
pod/argocd-application-controller-0 1/1    Running   0          34m
pod/argocd-dex-server-6c55787bc6-hrj4v 1/1    Running   0          34m
pod/argocd-redis-74d8c6db65-wzqfq 1/1    Running   0          34m
pod/argocd-repo-server-6c44847cf9-b9n6l 1/1    Running   0          34m
pod/argocd-server-67b65559fb-scgmp 1/1    Running   0          34m

```

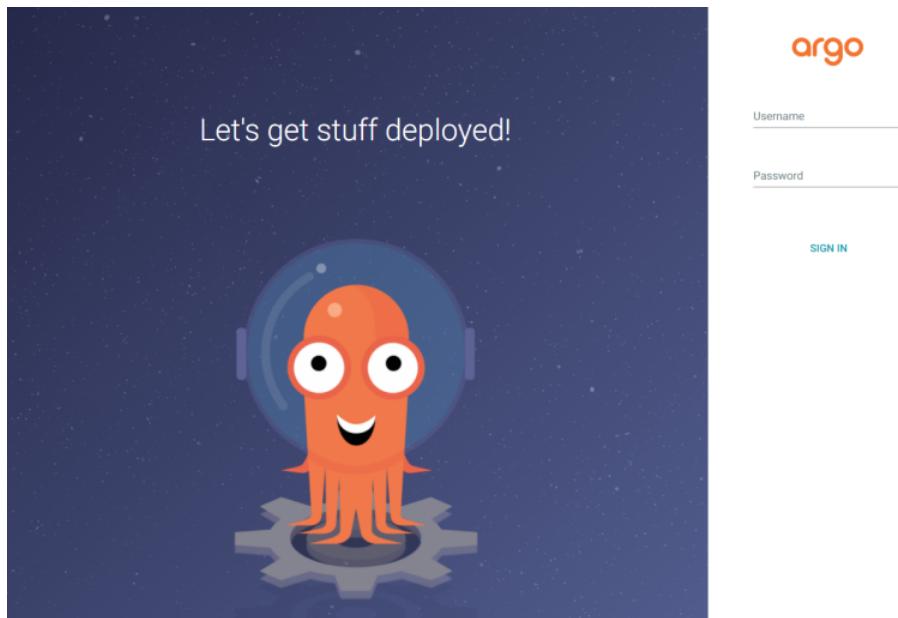
-

로드밸런서로 변경하고 서비스의 IP를 확인한다. 변경되는데는 1분 정도 소요된다.

```
// 출력  
NAME      TYPE      CLUSTER-IP    EXTERNAL-IP      PORT(S)          AGE  
argocd-server  LoadBalancer  10.8.9.12    <pending>        80:32524/TCP,443:31837/TCP  35m  
argocd-server  LoadBalancer  10.8.9.12    35.222.254.32   80:32524/TCP,443:31837/TCP  36m
```

로드밸런서로 할당 받은 IP로 https로 접속한다. <https://35.222.254.32/> 그리고 앞서 확인한 패스워드를 입력해 로그인을 수행한다.

```
username: admin  
password: <secret>을 통해 확인한 값>
```



로그인하면 다음과 같이 앱 창이 나타난다.

yaml 매니페스트 파일을 활용한 배포

여기서 New APP 버튼을 누르고 새로운 앱을 추가하자

GENERAL

Application Name
flask-example

Project
default

SYNC POLICY
Manual

SYNC OPTIONS

SKIP SCHEMA VALIDATION AUTO-CREATE NAMESPACE
 PRUNE LAST APPLY OUT OF SYNC ONLY

PRUNE PROPAGATION POLICY: foreground

REPLACE ⚠

- 매니페스트 파일의 소스의 위치를 지정한다. f
- lask-example-apps 프로젝트 링크를 전달하고 main 브랜치를 사용하도록 설정하자.
- Path는 배포할 yaml 파일이 있는 디렉토리를 지정한다. 우리의 프로젝트에는 flask-example-deploy 아래에 필요한 yaml이 구성되어 있다.

SOURCE

Repository URL
https://github.com/gasbugs/flask-example-apps

Revision
main

Path
flask-example-deploy

GIT ▾

Branches ▾

- 쿠버네티스 API 서버에 대한 정보를 입력한다.
- 우리가 구성한 argo는 쿠버네티스 내에 구성되었으므로 kube-apiserver에 대한 URL 정보를 도메인 주소로 입력할 수 있다.
- 네임스페이스는 애플리케이션을 배포할 공간을 의미한다. jenkins-ns에 배포를 진행하기 위해 다음과 같이 빈칸을 채운다.

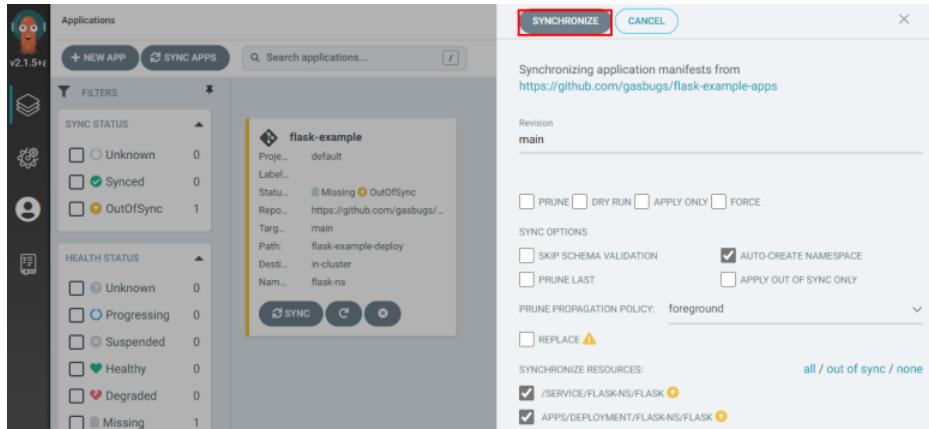
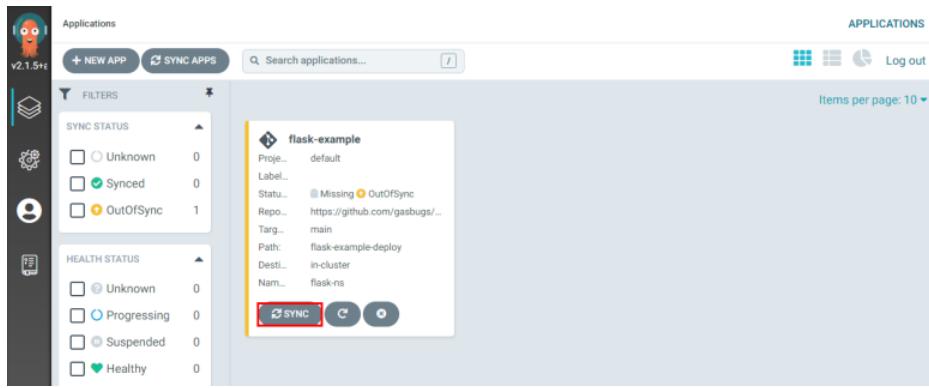
DESTINATION

Cluster URL
https://kubernetes.default.svc

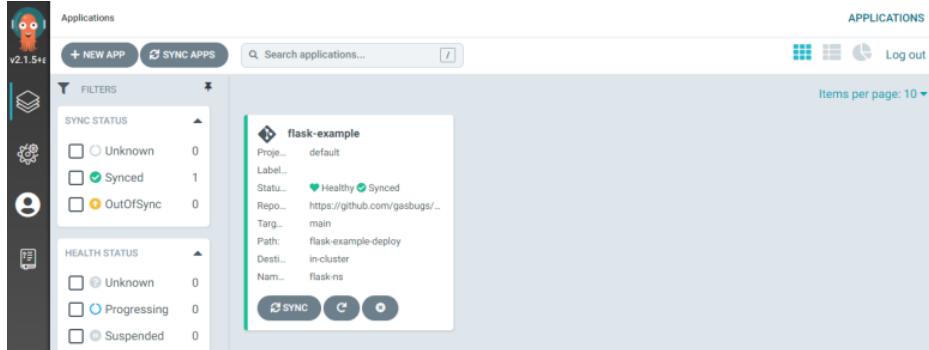
Namespace
flask-ns

URL ▾

모든 설정이 완료되면 상단에 CREATE 버튼을 누른다. 새로 생성된 app인 flask-example을 확인할 수 있다. SYNC 버튼을 눌러서 배포를 진행해보자.



시간이 조금 지나면 헬스체크와 싱크 상태를 알려주는 모습으로 변경된다.



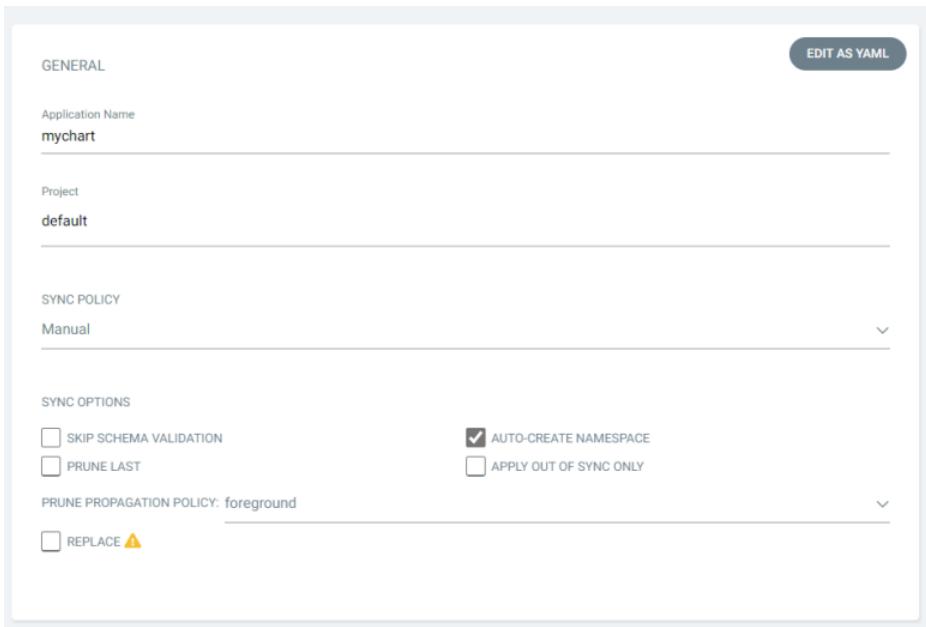
애플리케이션이 정상적으로 배포됐는지 다음 명령을 사용해 확인해보자.

```
$ kubectl get svc,deploy -n flask-ns
NAME           TYPE      CLUSTER-IP   EXTERNAL-IP   PORT(S)   AGE
service/flask   ClusterIP  10.8.10.189 <none>        80/TCP    2m6s

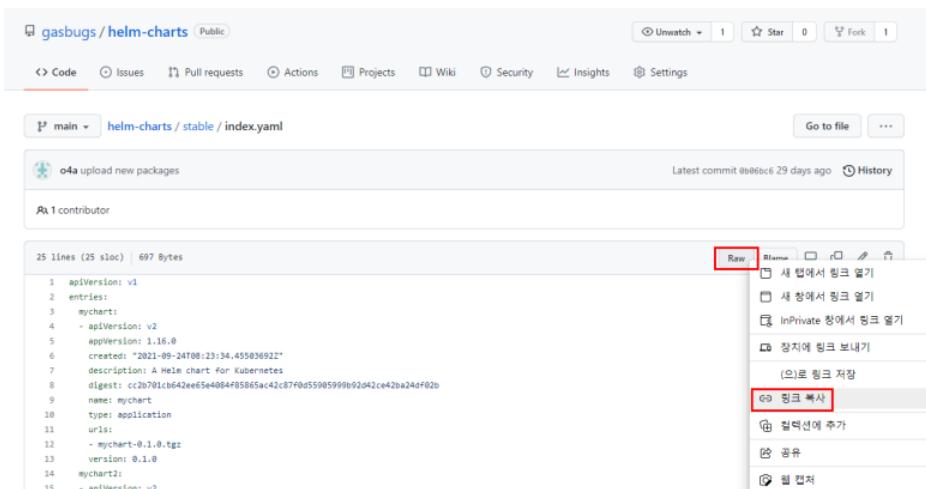
NAME          READY   UP-TO-DATE   AVAILABLE   AGE
deployment.apps/flask  1/1       1           1          2m6s
```

헬름차트를 활용한 배포

이번에는 헬름차트를 활용해 배포를 진행해보자. 앞서 구성한 방법과 동일하게 애플리케이션 이름과 몇 정보를 입력한다.



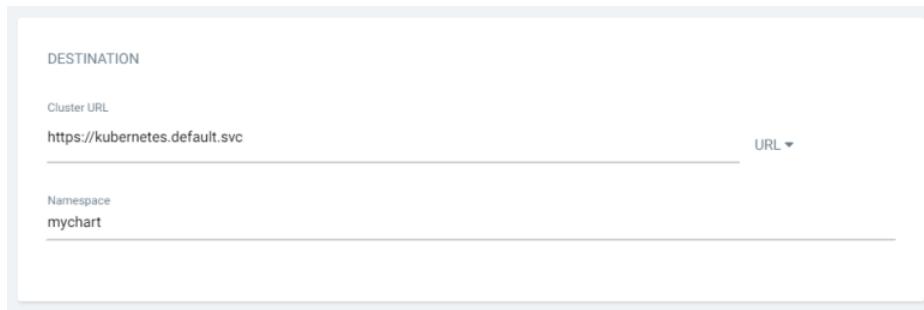
그리고 앞에서 소개한 helm-charts 레파지토리에서 index.yaml 파일을 찾자. 경로는 stable/index.yaml이다. 이 파일을 Raw 데이터로 가져오기 위해 마우스 우클릭 후 링크를 복사한다.



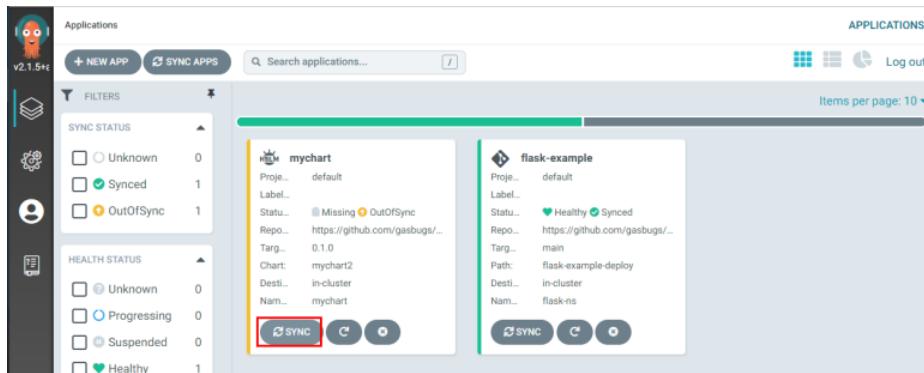
이 값을 SOURCE에 입력하되 index.yaml을 제거한다. 그리고 배포하려면 차트의 이름과 버전을 입력한다.



배포하려는 kube-apiserver와 네임스페이스를 입력하자.



모든 구성이 완료되었다면 Create를 누르고 SYNC - SYNCHRONIZE를 사용해 배포를 시작한다.



- 위와 같은 작업을 진행하면 최종적으로 다음과 같이 서비스 구성 확인할 수 있다.

Argo CD 최종 결과물

Project	Status	Repository	Target Revision	Path	Destination	Namespace	Created At
admin	Healthy Synced	https://github.com/wlwlsls/alback-helm-chart.git	main	admin	in-cluster	admin-ns	05/17/2023 02:32:39 (2 days ago)
concert	Healthy Synced	https://github.com/wlwlsls/alback-helm-chart.git	main	concert	in-cluster	concert-ns	05/17/2023 00:40:21 (2 days ago)
gateway	Healthy Synced	https://github.com/wlwlsls/alback-helm-chart.git	main	gateway	in-cluster	gateway-ns	05/15/2023 15:26:27 (4 days ago)
payment	Healthy Synced	https://github.com/wlwlsls/alback-helm-chart.git	main	payment	in-cluster	payment-ns	05/17/2023 02:52:30 (2 days ago)
user	Healthy Synced	https://github.com/wlwlsls/alback-helm-chart.git	main	user	in-cluster	user-ns	05/16/2023 03:02:48 (3 days ago)
concert	Missing OutOfSync	https://github.com/gasbugs/...	0.1.0	mychart2	in-cluster	mychart	

Monitoring : helm chart를 활용한 그라파나, 프로메테우스 배포

헬름 레파지토리 추가

클라우드 셸을 사용해 헬름 레파지토리를 추가하자. 헬름 v3를 사용한다.

```
helm repo add prometheus-community https://prometheus-community.github.io/helm-charts
helm repo add grafana https://grafana.github.io/helm-charts
helm repo update
```

그라파나와 프로메테우스 배포

헬름 배포를 위해 그라파나와 프로메테우스의 values.yaml을 구성할 디렉토리를 하나 구성한다.

```
mkdir grafana_prometheus
cd grafana_prometheus
```

다음 명령을 실행해 values-prometheus.yaml을 생성한다. pv를 구성하여 스토리지를 구성하고 15일간 데이터를 보존하도록 구성했다.

```
cat <<EOF > values-prometheus.yaml
server:
  enabled: true

  persistentVolume:
    enabled: true
    accessModes:
      - ReadWriteOnce
    mountPath: /data
    size: 100Gi
  replicaCount: 1

## Prometheus data retention period (default if not specified is 15 days)
##
  retention: "15d"
EOF
```

- 다음 명령을 실행해 values-grafana.yaml을 생성한다.
- pvc를 구성하여 스토리지를 구성하여 설정정보를 유지할 수 있도록 구성했다.
- 아이디 패스워드는 admin//test1234!234로 구성한다.
- 서비스가 생성될 때 접속하기 쉽도록 로드 밸런서로 구성했다.
- 실무에서는 모니터링 서비스가 외부로 노출되지 않도록 조심해야 한다.

```
cat << EOF > values-grafana.yaml
replicas: 1

service:
  type: LoadBalancer

persistence:
  type: pvc
  enabled: true
  # storageClassName: default
  accessModes:
    - ReadWriteOnce
  size: 10Gi
  # annotations: {}
  finalizers:
    - kubernetes.io/pvc-protection

# Administrator credentials when not using an existing secret (see below)
adminUser: admin
adminPassword: test1234!234
EOF
```

- 헬름으로 values 파일들을 사용해 배포를 시작한다.

```
kubectl create ns prometheus
helm install prometheus prometheus-community/prometheus -f values-prometheus.yaml -n prometheus
helm install grafana grafana/grafana -f values-grafana.yaml -n prometheus
```

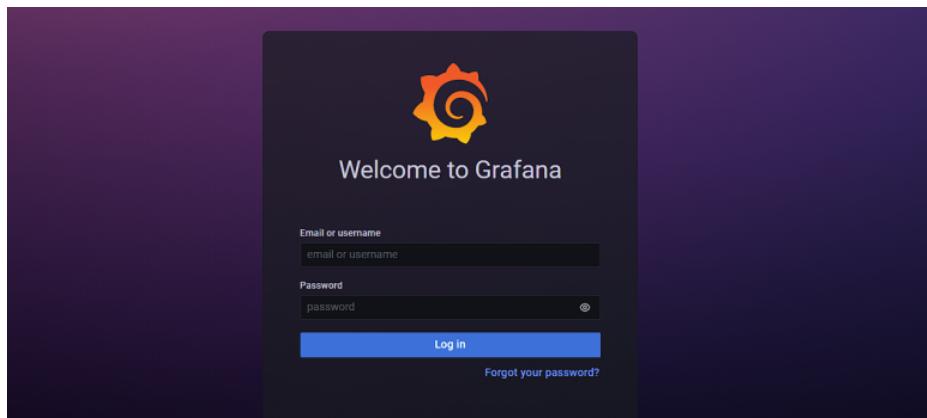
배포 확인

프로메테우스로 배포한 포드와 서비스가 잘 구성되었는지 확인한다.

```
$ kubectl get pod,svc -n prometheus
NAME                                         READY   STATUS    RESTARTS   AGE
pod/grafana-b64fbdc-b-qdxnm                 1/1     Running   0          49s
pod/prometheus-alertmanager-6755b9794f-thq5l   1/2     Running   0          57s
pod/prometheus-kube-state-metrics-696cf79768-xkjlk 1/1     Running   0          58s
pod/prometheus-node-exporter-bxvbw           1/1     Running   0          58s
pod/prometheus-node-exporter-dlmnl           1/1     Running   0          58s
pod/prometheus-node-exporter-qgjpc           1/1     Running   0          58s
pod/prometheus-pushgateway-898d5bdb9-7bh7h   1/1     Running   0          58s
pod/prometheus-server-c68ccc7ff-54jzw        2/2     Running   0          58s

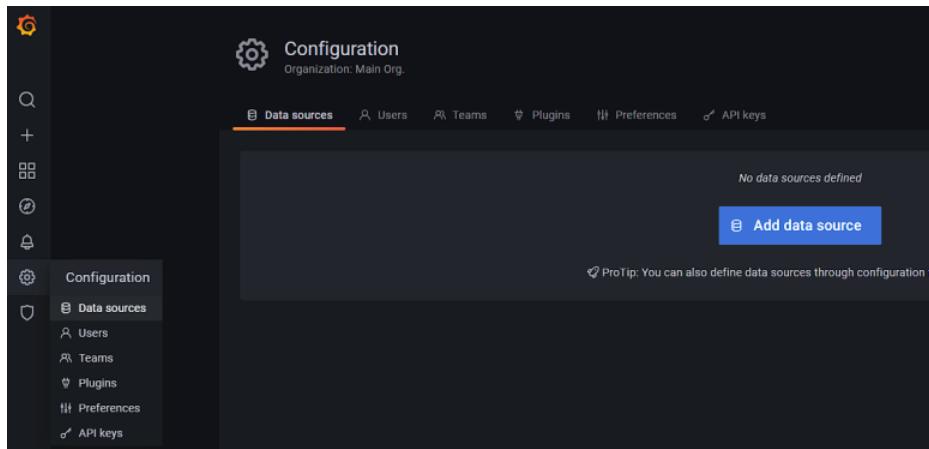
NAME                TYPE      CLUSTER-IP      EXTERNAL-IP      PORT(S)      AGE
service/grafana     LoadBalancer  10.8.0.77     34.70.251.10   80:31086/TCP  51s
service/prometheus-alertmanager   ClusterIP   10.8.14.141   <none>        80/TCP       59s
service/prometheus-kube-state-metrics  ClusterIP   10.8.6.161    <none>        8080/TCP    59s
service/prometheus-node-exporter      ClusterIP   None          <none>        9100/TCP    59s
service/prometheus-pushgateway     ClusterIP   10.8.13.17   <none>        9091/TCP    59s
service/prometheus-server          ClusterIP   10.8.4.29     <none>        80/TCP       59s
```

외부 IP로 노출된 grafana의 IP로 접속을 수행한다. 아이디와 패스워드는 admin//test1234!234다.

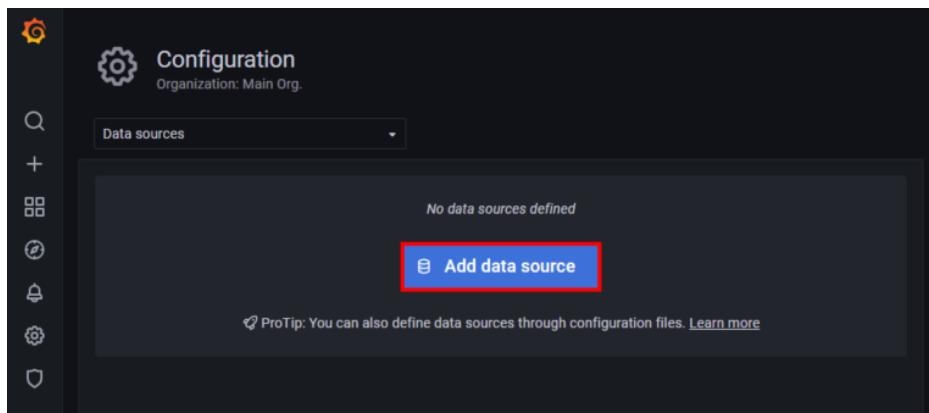


프로메테우스 데이터를 그라파나로 가져오기

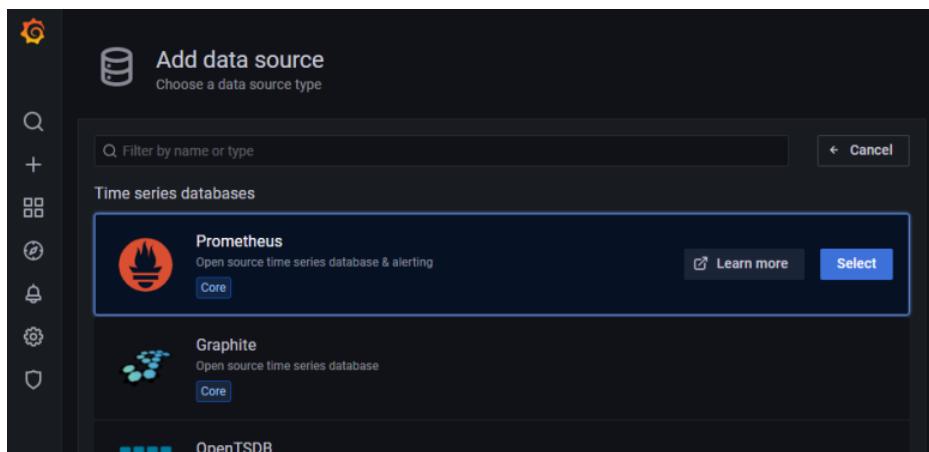
프로메테우스의 데이터를 가져오기 Configuration의 Data sources로 접근한다.



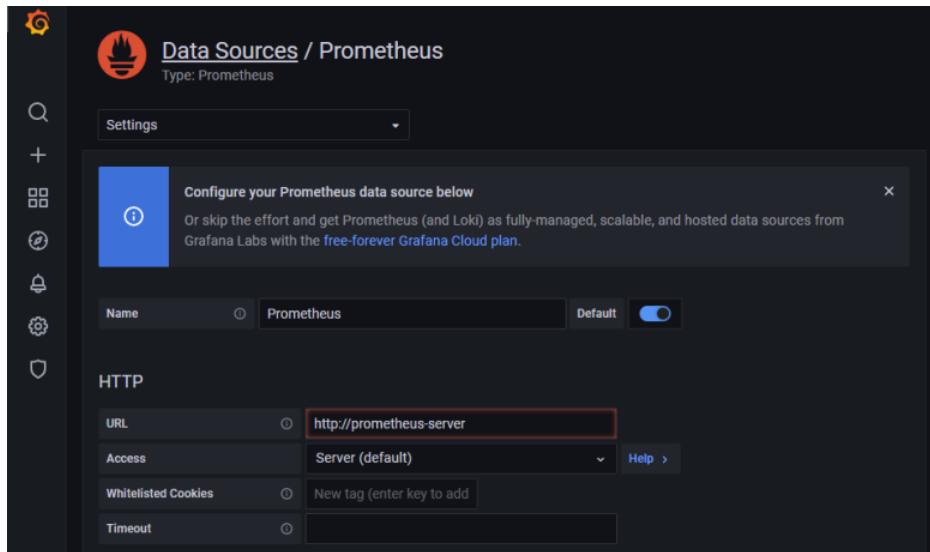
데이터 추가를 위해 Add data source를 클릭하자.



데이터 소스 타입을 Prometheus로 선택한다.

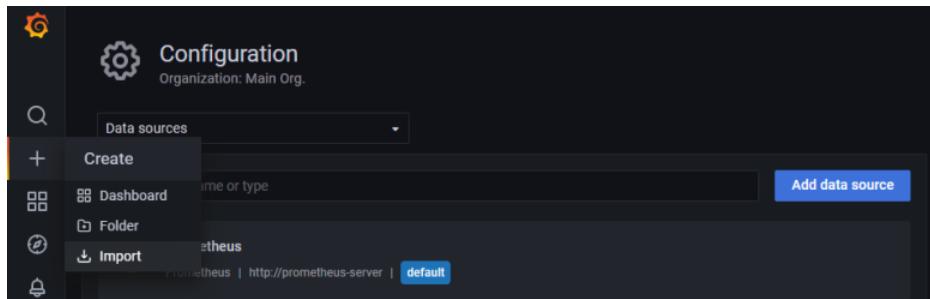


HTTP 서버에 대한 URL 정보에 도메인 정보를 입력하자. 프로메테우스의 도메인 이름은 서비스를 확인하면 된다. 앞서 확인한 정보에서 service/prometheus-server 이름을 확인할 수 있다. 여기서는 별도의 인증정보가 필요 없으므로 바로 save & test 버튼을 클릭한다.



대시보드 구성하기

대시보드를 구성하기 위해 +버튼 (Create)의 Import로 들어간다. Import는 외부에 사용자들이 미리구성해놓은 좋은 다양한 대시보드를 간단히 다운로드 받아서 설정할 수 있다.



- 그라파나 사이트에 다양한 모양을 가진 대시보드들이 올라와 있다. 원하는 것을 찾고 ID를 가져와서 입력하면 된다.



All dashboards > Kubernetes cluster monitoring (via Prometheus)



Kubernetes cluster monitoring (via Prometheus)

by Instrumentisto Team

Monitors Kubernetes cluster using Prometheus. Shows overall cluster CPU / Memory / Filesystem usage as well as individual pod, containers, systemd services statistics. Uses cAdvisor metrics only.

Last updated: 8 months ago

Start with Grafana Cloud and the new FREE tier: Includes 10K series Prometheus or Graphite Metrics and 50gb Loki Logs

Downloads: 182409
Reviews: 7
★★★★★
[Add your review!](#)

[Overview](#)
[Revisions](#)
[Reviews](#)







Get this dashboard:

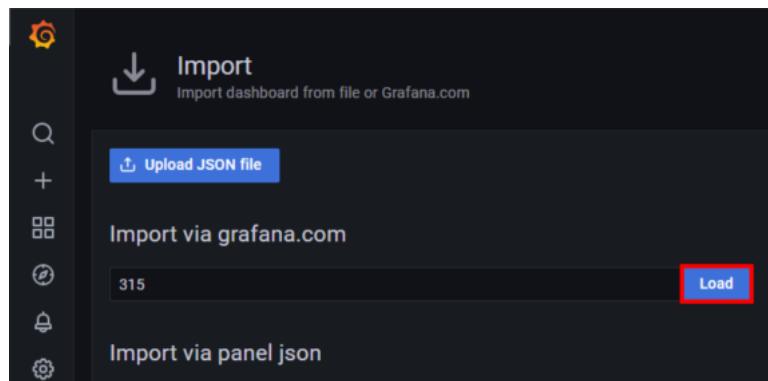
315

[Copy ID to Clipboard](#)

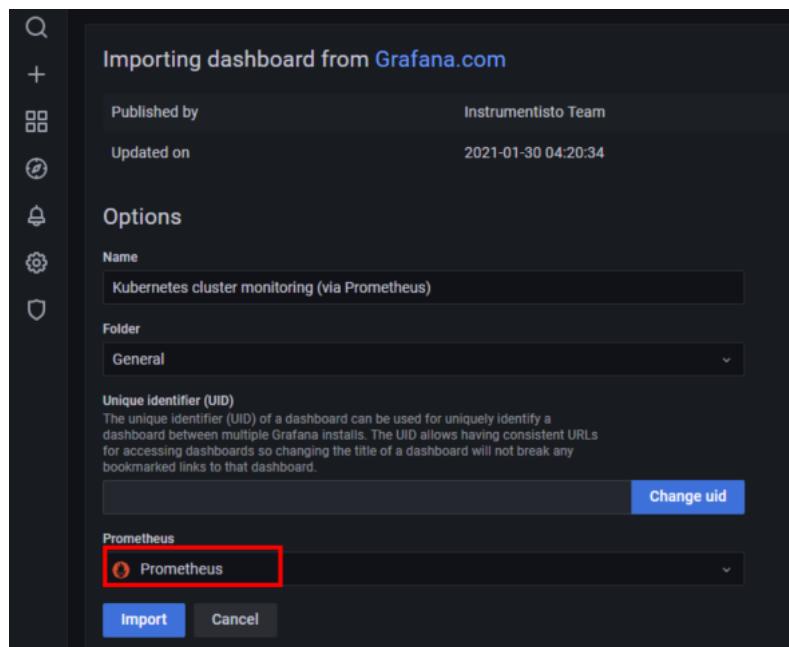
Download JSON
How do I import this dashboard?

Dependencies:
GRAFANA 3.1.1

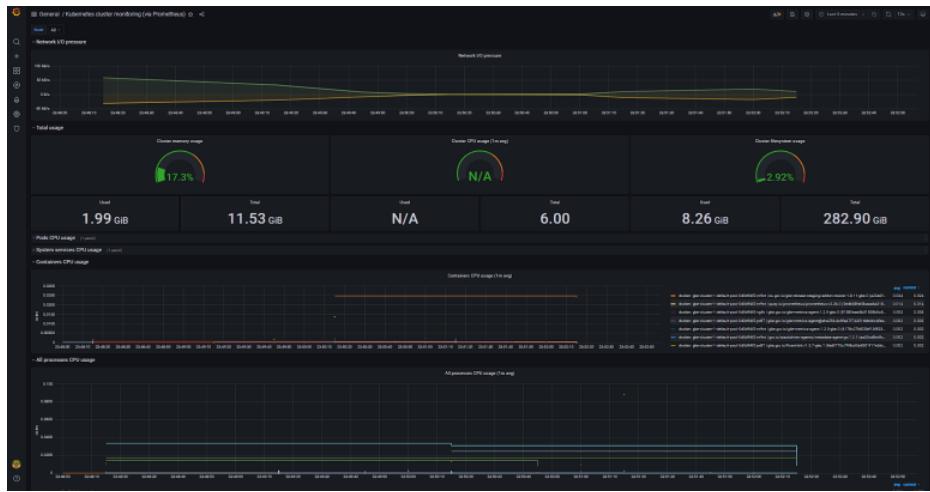
복사한 ID를 입력하고 Load 버튼을 누르면 바로 구성이 시작된다.



남은 정보를 마저 설정하고 Import를 누른다.



구성을 완료하면 다음과 같은 쿠버네티스 자원 사용 현황을 대시보드로 파악할 수 있다.



모니터링 최종 결과물

