

포팅매뉴얼_A610

목차

1. 포팅매뉴얼_NGINX서버
2. 포팅매뉴얼_메인서버
3. 포팅매뉴얼_Prometheus
4. 포팅매뉴얼_Grafana
5. 포팅매뉴얼_nGrinder
6. 포팅매뉴얼_ChatBot
7. 포팅매뉴얼_vector DB
8. 포팅매뉴얼_DB 서버
9. 포팅매뉴얼_ChromeDriver
10. 포팅매뉴얼_프론트

포팅매뉴얼_NGINX서버

1. 서버 시간 설정

```
#현재 시간 설정 확인
date

#목록에 서울 시간 있는지 확인
timedatectl list-timezones | grep Seoul

#서울 시간으로 변경
sudo timedatectl set-timezone Asia/Seoul
```

도메인 구매

서비스 신청



1. 신청 정보 입력



2. 서비스 비용 결제



3. 정보 확인

신청 정보

도메인	등록 비용 X 기간: 1년 ▼
benepick.shop	500원 x 1년 ▼ = 500원

소유자 정보 ①

소유자 목록

별칭	회원정보	회원 구분	<input checked="" type="radio"/> 개인 <input type="radio"/> 기관
소유자명(한글)	박시균	소유자명(영문)	ParkSiGyun
소유자 인증용 필수 정보 ①	이메일 psg980331@naver.com	인증 완료	이메일 재입력
	휴대전화 010 - 9950 - 9587		
전화번호	02 - -		
우편번호		검색	
주소(한글)			
주소(영문)			

결제 금액 계산	
항목	비용
도메인(1)	49,000원
L98% 할인	-48,500원
부가세	50원
결제 예상 금액 550원	
다음 단계 >	
< 이전 단계	
권장사 추천	
가비아 도메인	

신청 정보

도메인 등록	benepick.shop/1년
--------	------------------

* 일부 TLD의 경우 결제 완료 이후에도 등록 또는 만기일 연장이 실시간으로 진행되지 않을 수 있습니다.

관리 정보

소유자 정보			
소유자명(한글/영문)	박시균(ParkSiGyun)	이메일	psg980331@naver.com
전화번호	010-9950-9587	휴대전화	010-9950-9587
관리자 정보			
관리자명(한글/영문)	박시균(ParkSiGyun)	이메일	psg980331@naver.com
전화번호	010-9950-9587	휴대전화	010-9950-9587
네임서버	ns.gabia.co.kr 43,201,170,100 ns1.gabia.co.kr 20,200,205,248 ns.gabia.net 121,78,117,39		



가비아를 통해 도메인 구매 (benepick.shop)

Route53 & 가비아 설정

호스팅 영역 구성

호스팅 영역은 example.com 같은 도메인과 관련 하위 도메인에 대한 트래픽을 라우팅하는 방식에 대한 정보를 포함하는 컨테이너입니다.

도메인 이름 | 정보

트래픽을 라우팅할 도메인의 이름입니다.

benepick.shop

유효한 문자: a-z, 0-9 및 ! " # \$ % & ' () * + , - / : ; < = > ? @ [\] ^ _ ` { | } . ~

설명 - 선택 사항 | 정보

이 값을 사용하면 이름이 동일한 호스팅 영역을 구별할 수 있습니다.

호스팅 영역이 사용되는 경우...

설명은 최대 256자입니다. 0/256

유형 | 정보

유형은 인터넷 또는 Amazon VPC에서 트래픽을 라우팅할지 여부를 가리킵니다.

☒ 퍼블릭 호스팅 영역

퍼블릭 호스팅 영역은 인터넷에서 트래픽을 라우팅하는 방식을 결정합니다.

☐ 프라이빗 호스팅 영역

프라이빗 호스팅 영역은 Amazon VPC 내에서 트래픽을 라우팅하는 방식을 결정합니다.

퍼블릭 benepick.shop 정보

영역 삭제

레코드 테스트

쿼리 로깅 구성

▶ 호스팅 영역 세부 정보

호스팅 영역 편집

레코드(2)

DNSSEC 서명

호스팅 영역 태그(0)

레코드 (2) 정보

Automatic 모드는 최상의 필터 결과에 최적화된 현재 검색 동작입니다. 모드를 변경하려면 설정(settings)으로 이동합니다.

<input type="checkbox"/>	레코드 ... ▼	유형 ▼	라우팅 ... ▼	차별... ▼	별칭 ▼	값/트래픽 라우팅 대상 ▼	TTL(초) ▼	상태 확... ▼
<input type="checkbox"/>	benepick....	NS	단순	-	아니요	ns-303.awsdns-37.com. ns-1229.awsdns-25.org. ns-532.awsdns-02.net. ns-2019.awsdns-60.co.uk.	172800	-
<input type="checkbox"/>	benepick....	SOA	단순	-	아니요	ns-303.awsdns-37.com. awsd...	900	-

benepick.shop ▲

네임서버 목록

구분	호스트명	구분	호스트명
1차	ns-303.awsdns-37.com	2차	ns-1229.awsdns-25.org
3차	ns-532.awsdns-02.net	4차	ns-2019.awsdns-60.co.uk

- 네임서버는 IP(숫자)를 제외한 호스트명만 입력합니다. (예. ns.gabia.co.kr)
- 네임서버 값을 복사해서 입력하는 경우, 공란이 포함되지 않도록 주의하시기 바랍니다.
- 각 도메인마다 네임서버 최소/최대 값이 다릅니다.
- 따라서 여러 개의 네임 서버를 입력하여도 도메인의 허용된 개수에 따라 적용됩니다.
- 가비아 네임서버는 DNSSEC을 지원하지 않습니다. DNSSEC이 설정된 도메인을 가비아 네임서버로 변경하시는 경우, 웹사이트 접속(리출방)이 제한됩니다.

단순 레코드 정의



레코드 이름 | 정보

트래픽을 하위 도메인으로 라우팅하려면 하위 도메인 이름을 입력합니다. 예를 들어, 트래픽을 blog.example.com으로 라우팅하려면 **blog**을(를) 입력합니다. 이 필드를 비워 두면 기본 레코드 이름이 도메인 이름입니다.

subdomain

benepick.shop

루트 도메인에 대한 레코드를 생성하려면 비워 둡니다.

레코드 유형 | 정보

Route 53가 DNS 쿼리에 대한 응답으로 반환하는 값의 형식을 결정하는 레코드의 DNS 유형입니다.

A - IPv4 주소 및 일부 AWS 리소스로 트래픽 라우팅 ▼

EC2, API Gateway, Amazon VPC, CloudFront, Elastic Beanstalk, ELB 또는 S3에서 AWS 리소스로 트래픽을 라우팅할 때 선택합니다. 예: 192.0.2.44.

값/트래픽 라우팅 대상 | 정보

선택하는 옵션에 따라 Route 53이 DNS 쿼리에 응답하는 방법이 결정됩니다. 대부분의 옵션의 경우, 인터넷 트래픽을 라우팅할 위치를 지정합니다.

엔드포인트 선택 ▼

43.202.115.193

별도의 줄에 여러 값을 입력합니다.

TTL(초) | 정보

DNS 해석기 및 웹 브라우저가 이 레코드의 설정을 캐시하는 시간(초)입니다("TTL"은 "Time To Live"를 의미함).

300

1분

1시간

1일

취소

단순 레코드 정의



1. Route53 호스팅 영역 생성
2. 생성된 호스팅 영역의 라우팅 대상의 4개를 복사하여 가비아의 네임서버에 등록해준다.
3. 새로운 레코드를 생성해준다.

2. Nginx 설치

```
sudo apt-get update
sudo apt-get install nginx

#실행전 80번 포트를 사용하는 nginx를 위해 방화벽 허용
sudo ufw allow 22/tcp
sudo ufw allow 80/tcp
sudo ufw allow 443/tcp
sudo ufw enable

# ec2 인바운드 설정 80,443 포트 허용 설정

#nginx 실행
sudo systemctl start nginx
sudo systemctl stop nginx
sudo systemctl status nginx

#인증서 발급
sudo snap install certbot --classic
sudo certbot --nginx
#이메일 쓰고 agree
#뉴스레터 no
#도메인 입력
#인증서 발급 성공 , 인증서 보관하기
```

```
# cd etc/nginx/conf.d
# sudo vim default.conf
# etc/nginx/conf.d/default.conf

upstream backend{
    # least_conn;
    # 로드밸런싱 디폴트 알고리즘 : Round Robin
    server 15.164.146.106:8083;
    server 3.36.207.27:8083;
}

upstream bank {
    server 15.164.146.106:8084;
    server 3.36.207.27:8084;
}

upstream gpt {
    server 15.164.146.106:3001;
    server 3.36.207.27:3001;
}

server{
    # 80 port로 서버 오픈
    listen 80;
    #IPv6 주소에서 들어오는 요청을 처리
    listen [::]:80;
    #서버 이름
    server_name benepick.shop;
    #HTTP 요청을 받으면 모두 HTTPS로 리디렉션(301 Redirect)
    return 308 https://$server_name$request_uri;
}

server{
    include /etc/nginx/conf.d/service-port.inc;
    include /etc/nginx/conf.d/bank-port.inc;
    #포트 443과 IPv4 주소, 그리고 포트 443과 IPv6 주소에서 들어오는 요청을 처리
    listen 443 ssl;
    listen [::]:443 ssl;

    #서버 이름
    server_name benepick.shop;

    #HTTPS 요청을 받으면 /var/www/html 디렉토리에 위치한 정적 파일들을 서비스
    #root /var/www/html;
    #index index.html index.htm index.nginx-debian.html;

    #2가지 키가 발급
    ssl_certificate /etc/letsencrypt/live/benepick.shop/fullchain.pem; # managed by Certbot
```

```

ssl_certificate_key /etc/letsencrypt/live/benepick.shop/privkey.pem; # managed by Certbot

location / {
    proxy_pass http://backend; # 설정한 이름으로 요청 보내기
}

location /bank {
    proxy_pass http://bank;
}

location /gpt {
    proxy_pass http://gpt;
}

proxy_set_header Host $http_host;
proxy_set_header Connection $http_connection;
proxy_set_header X-Real-IP $remote_addr;
proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
proxy_set_header X-Forwarded-Proto $scheme;
add_header 'Access-Control-Allow-Origin' '*';
}

```



설정후 nginx 재실행
systemctl restart nginx

포팅매뉴얼_메인서버

1. 서버 시간 설정

```

#현재 시간 설정 확인
date

#목록에 서울 시간 있는지 확인
timedatectl list-timezones | grep Seoul

#서울 시간으로 변경
sudo timedatectl set-timezone Asia/Seoul

```

2. 도커 설치

```

# 우분투 시스템 패키지 업데이트
sudo apt update

# 필요 패키지 설치
sudo apt-get install apt-transport-https ca-certificates curl gnupg-agent software-properties-common

# Docker 공식 GPG 키 추가
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -

# Docker 공식 apt 저장소 추가
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu $(lsb_release -cs) stable"

# 시스템 패키지 업데이트
sudo apt-get update

# Docker 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io

# Docker 설치 확인
sudo systemctl status docker

```

3. 노드 익스포터 설치 및 실행

```
# 압축파일 설치
wget https://github.com/prometheus/node_exporter/releases/download/v1.6.1/node_exporter-1.6.1.linux-amd64.tar.gz
# 압축 해제
tar xzvf node_exporter-1.6.1.linux-amd64.tar.gz
# 노드 익스포터 백그라운드 실행
./node_exporter/node_exporter &
```

4. Jenkins 설정

Jenkins 컨테이너 설치 및 실행

```
$ docker run -d --name jenkins --restart=on-failure \
-p 8080:8080 \
-v /var/jenkins_home:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-e TZ=Asia/Seoul \
-u root \
jenkins/jenkins
```

▼ 명령어 옵션

- **-restart** -> on-failure 옵션은 비정상 종료시 컨테이너를 재실행합니다.
- **p** -> 외부 접속을 위해 호스트의 8080 포트를 바인딩 해주었습니다.
- **v** -> 호스트의 /var/jenkins 디렉토리를 호스트 볼륨으로 설정하여 jenkins 컨테이너의 home 디렉토리에 마운트시켰습니다.

docker.sock 파일은 도커 데몬과 통신할 수 있는 소켓 파일입니다. docker.sock 파일을 컨테이너에 마운트시켜서 도커 명령을 실행할 수 있게 해줍니다. 이러한 방식을 dood(docker out of docker)라고 합니다.

- **e** -> 젠킨스의 timezone을 KST 기준으로 설정해줍니다.
- **u** -> 추후 권한 문제가 발생할 수 있기 때문에 user 옵션을 root 사용자로 주었습니다.

```
ubuntu@ip-172-31-30-87:~$ sudo docker run -d --name jenkins --restart=on-failure -p 8080:8080 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -e TZ=Asia/Seoul -u root jenkins/jenkins
Unable to find image 'jenkins/jenkins:latest' locally
latest: Pulling from jenkins/jenkins
012c0b3e998c: Pull complete
08d5bc5026df: Extracting 27.3MB/61.32MB
ced1909dae03: Download complete
15e27b0be22f: Download complete
3f68c4dffe66: Download complete
e5ff73647592: Download complete
a96a1d982b84: Download complete
b24a00d8fb00: Download complete
4099a34af478: Download complete
daa8ff635634: Download complete
086249825169: Download complete
15a814ee2c77: Download complete
12b571a61f13: Download complete
```

초기 비밀번호 입력

```
$ docker exec -it jenkins /bin/bash
$ cat /var/jenkins_home/secrets/initialAdminPassword
```

```
ubuntu@ip-172-31-30-87:~$ sudo docker exec -it jenkins /bin/bash
root@e3d448774061:/# cat /var/jenkins_home/secrets/initialAdminPassword
af564373e6c444148b3c0788031bcbaa
root@e3d448774061:/#
```

▼ 젠킨스 깃랩 연동

1. 초기 비밀번호 입력

Getting Started

Unlock Jenkins

To ensure Jenkins is securely set up by the administrator, a password has been written to the log ([not sure where to find it?](#)) and this file on the server:

```
/var/jenkins_home/secrets/initialAdminPassword
```

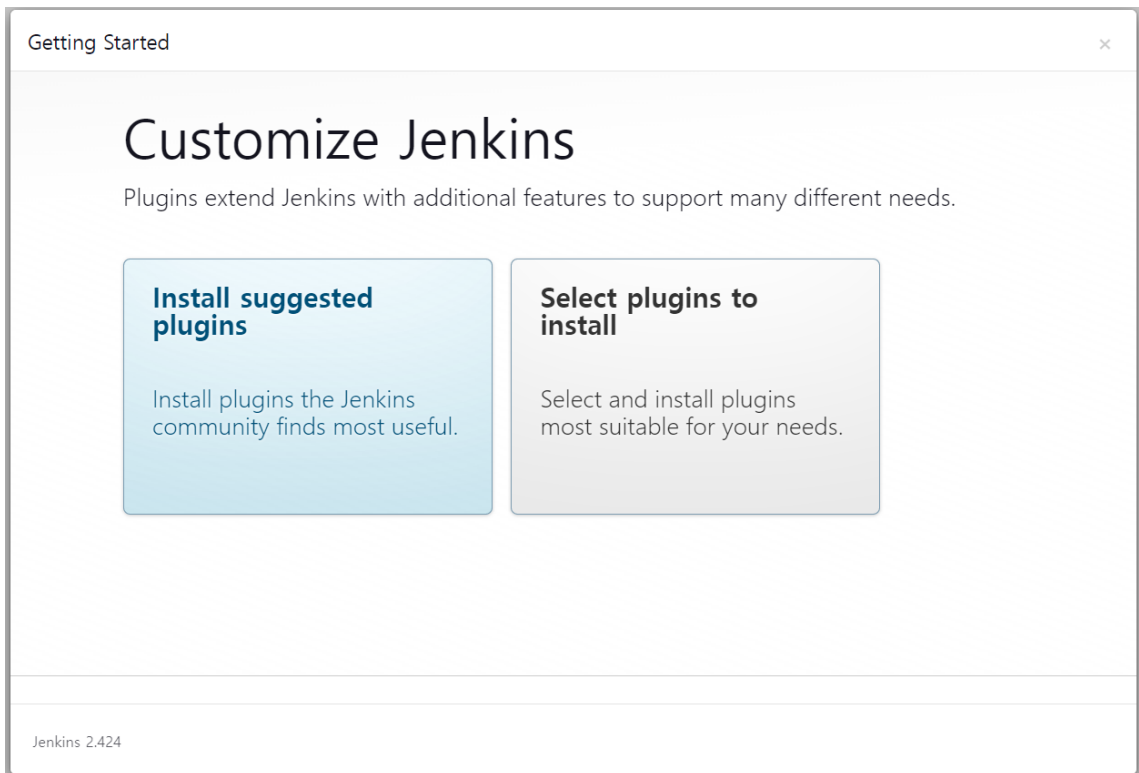
Please copy the password from either location and paste it below.

Administrator password

.....

Continue

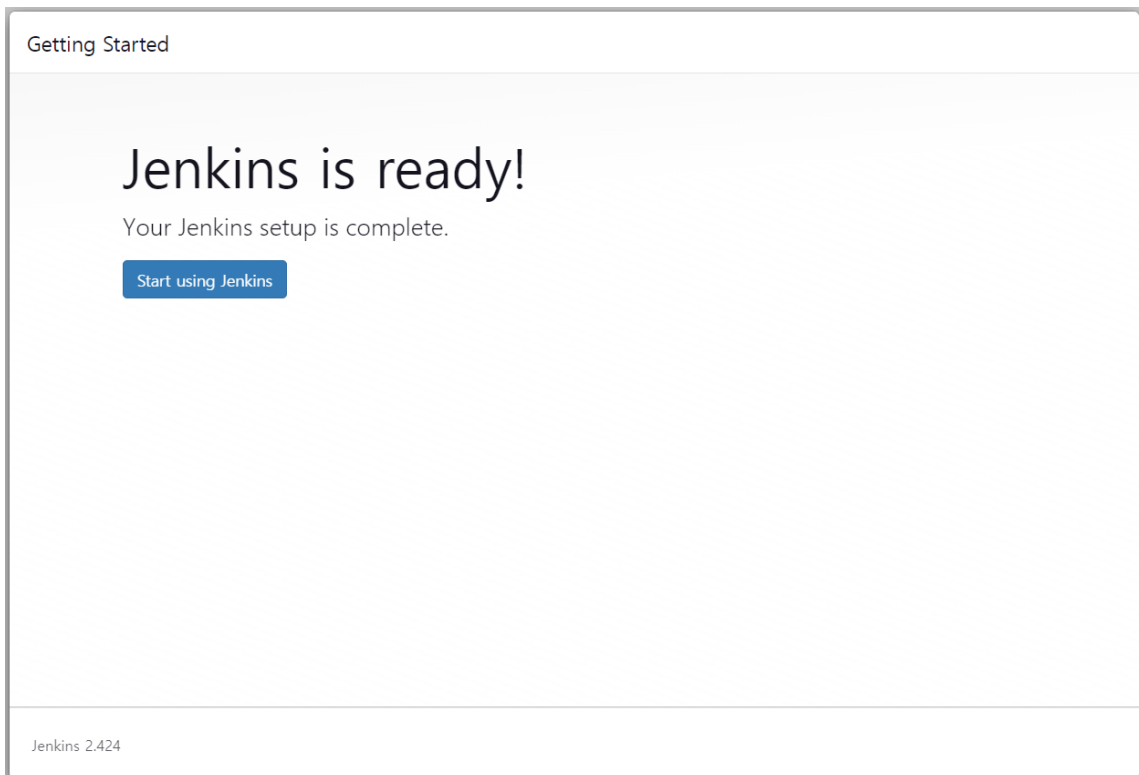
2. install suggested plugins 시작



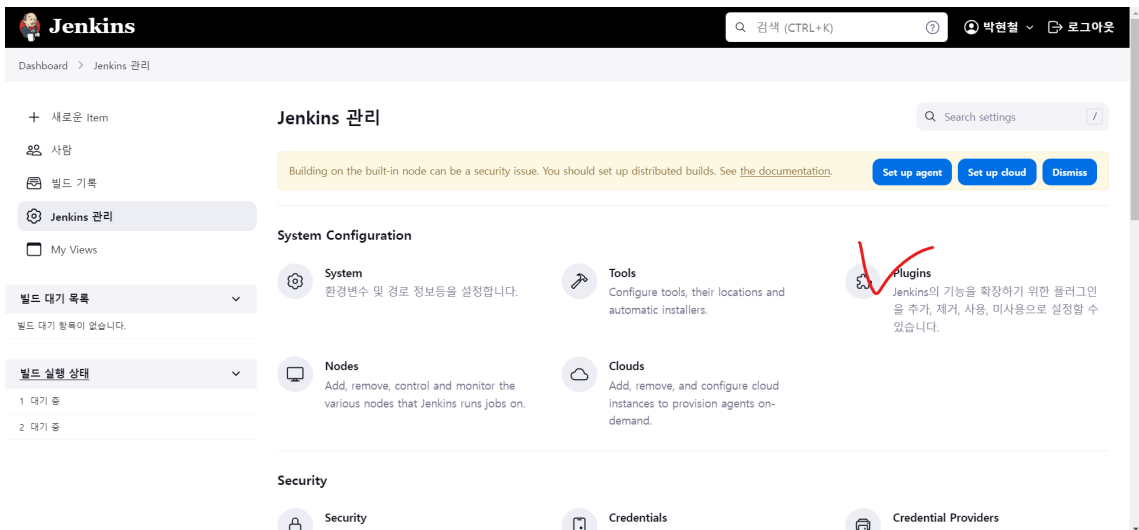
3. 유저 계정 생성

This screenshot shows the 'Create First Admin User' window in the Jenkins installation process. The window has a title bar 'Getting Started'. The main heading is 'Create First Admin User'. Below the heading, there are five input fields with labels in Korean: '계정명' (Username) with the value 'benepick', '암호' (Password) with masked characters, '암호 확인' (Confirm Password) with masked characters, '이름' (Name) with the value '박현철', and '이메일 주소' (Email address). At the bottom left, it says 'Jenkins 2.424'. At the bottom right, there are two buttons: 'Skip and continue as admin' and 'Save and Continue'.

4. 젠킨스 시작



5. 젠킨스 플러그인 설치



▼ 설치 플러그인

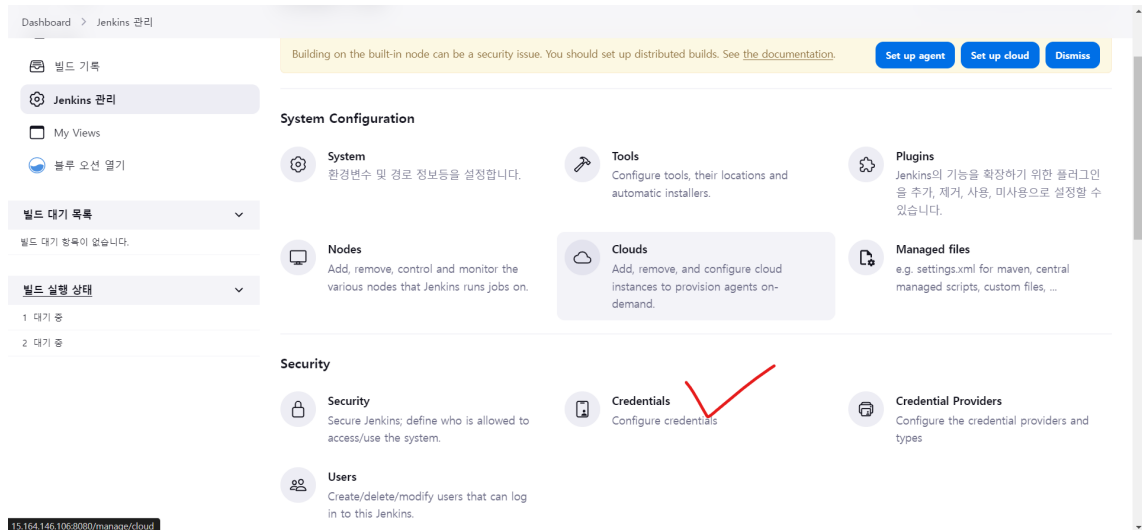
1. Bitbucket Pipeline for Blue Ocean
2. Dashboard for Blue Ocean
3. Personalization for Blue Ocean
4. Display URL for Blue Ocean
5. Server Sent Events (SSE) Gateway
6. Events API for Blue Ocean
7. Blue Ocean Pipeline Editor
8. i18n for Blue Ocean
9. Autofavorite for Blue Ocean
10. Blue Ocean
11. NodeJS
12. GitLab
13. Generic Webhook Trigger
14. Gitlab Authentication
15. Gitlab API
16. Gitlab Branch Source
17. Gitlab Merge Request Builder

```

18. Config File Provider
19. Docker
20. Docker Pipeline
21. docker-build-step
22. git parameter

```

6. 젠킨스 인증키 생성




The screenshot shows the Jenkins Dashboard with the 'Jenkins 관리' (Jenkins Management) tab selected. The 'System Configuration' section is visible, containing links for System, Tools, Plugins, Nodes, Clouds, and Managed files. The 'Security' section is also visible, containing links for Security, Credentials (marked with a red checkmark), and Credential Providers. The 'Users' link is also present. The 'Credentials' link is highlighted with a red checkmark.

Credentials

T	P	Store ↓	Domain	ID	Name
---	---	---------	--------	----	------

Stores scoped to Jenkins

P	Store ↓	Domains
	System	(global)

아이콘: S M L

Kind

Username with password

Scope ?

Global (Jenkins, nodes, items, all child items, etc)

Username ?

hyunchul1998

☐ Treat username as secret ?

Password ?

.....

ID ?

benepick

Description ?

Create

▼ 입력 정보

- kind : Username with password
- Scope : Global (Jenkins, nodes, items, all child items, etc)


- Username : GitLab 사용중인 아이디 (중요)★
- Password : GitLab 에서 발급받은 토큰 값
- ID : 사용할 ID

7. item 이름 입력 후, 프로젝트 생성

Enter an item name


benepick

» A job already exists with the name 'benepick'




Freestyle project

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




Maven project

Maven 프로젝트를 빌드합니다. Jenkins은 POM 파일의 이점을 가지고 있고 급격히 설정을 줄입니다.




Pipeline

Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.



Multi-configuration project

다양한 환경에서의 테스트, 플레폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.



Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder

8. 소스코드 관리에 연동할 GitLab https 주소 및 생성한 인증키 입력

소스 코드 관리

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://lab.ssafy.com/s09-fintech-finance-sub2/S09P22A610.git

Credentials ?

hyunchul1998/*****

+ Add ▾

고급 ▾

Add Repository

저장

Apply

9. 젠킨스 아이템 깃랩에 등록

빌드 유발

- ☐ 빌드를 원격으로 유발 (예: 스크립트 사용) ?
- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: <http://15.164.146.106:8080/project/benepick> ?

Enabled GitLab triggers

- ☒ Push Events ?
- ☐ Push Events in case of branch delete ?
- ☒ Opened Merge Request Events ?
- ☐ Build only if new commits were pushed to Merge Request ?
- ☐ Accepted Merge Request Events ?
- ☐ Closed Merge Request Events ?

Rebuild open Merge Requests ?

Never

- ☒ Approved Merge Requests (EE-only) ?
- ☒ Comments ?

저장

Apply

Secret token ?

be9d27b1eee842d9e2b61830b5e8b073

Generate

고급 → 시크릿 토큰 발급

Search page

Webhooks

Webhooks enable you to send notifications to web applications in response to events in a group or project. We recommend using an [integration](#) in preference to a webhook.

URL

<http://15.164.146.106:8080/project/benepick>

URL must be percent-encoded if it contains one or more special characters.

☒ Show full URL

☐ Mask portions of URL

Do not show sensitive data such as tokens in the UI.

Secret token

.....

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

깃랩 webhooks → url, 시크릿 토큰 입력

10. 소스코드 관리 → branch specifier에 GitLab Hooks 브랜치 설정 (*develop)

Branches to build ?

Branch Specifier (blank for 'any') ?

`${GIT_BRANCH}`

Branch Specifier (blank for 'any') ?

`*/develop`

젠킨스 스크립트

```
EXIST_BLUE=$(docker-compose -p benepick-blue -f /var/jenkins_home/workspace/benepick/docker-compose/docker-compose.blue.yaml ps | grep

if [ -z "$EXIST_BLUE" ]; then
    # blue 컨테이너가 실행 중이지 않을 경우 blue 컨테이너를 시작.
    docker-compose -p benepick-blue -f /var/jenkins_home/workspace/benepick/docker-compose/docker-compose.blue.yaml up --build -d
    BEFORE_COLOR="green"
    AFTER_COLOR="blue"
    BEFORE_MAIN_PORT=8083
    BEFORE_BANK_PORT=8084
    AFTER_MAIN_PORT=8081
    AFTER_BANK_PORT=8082
else
    # blue 컨테이너가 실행 중이면 green 컨테이너를 시작
    docker-compose -p benepick-green -f /var/jenkins_home/workspace/benepick/docker-compose/docker-compose.green.yaml up --build -d
    BEFORE_COLOR="blue"
    AFTER_COLOR="green"
    BEFORE_MAIN_PORT=8081
    BEFORE_BANK_PORT=8082
    AFTER_MAIN_PORT=8083
    AFTER_BANK_PORT=8084
fi

echo "${AFTER_COLOR} server up(main_port:${AFTER_MAIN_PORT}, bank_port:${AFTER_BANK_PORT})"

# 2
for cnt in {1..10}
do
    echo "메인 서버 응답 확인중(${cnt}/10)";
    UP=$(curl -s http://15.164.146.106:${AFTER_MAIN_PORT}/api/health-check || true)
    if [ -z "${UP}" ]
    then
        sleep 10
        continue
    else
        break
    fi
done

if [ $cnt -eq 10 ]
then
    echo "메인 서버가 정상적으로 구동되지 않았습니다."
    exit 1
fi

# 2
for cnt in {1..10}
do
    echo "뱅크 서버 응답 확인중(${cnt}/10)";
    UP=$(curl -s http://15.164.146.106:${AFTER_BANK_PORT}/api/health-check || true)
    if [ -z "${UP}" ]
    then
        sleep 10
        continue
    else
        break
    fi
done

if [ $cnt -eq 10 ]
then
    echo "뱅크 서버가 정상적으로 구동되지 않았습니다."
    exit 1
fi

# Nginx 포트 동적 변경
# Nginx 설정 파일 경로
NGINX_CONFIG_FILE="/etc/nginx/conf.d/default.conf"

# 원격 EC2 인스턴스의 IP 주소 또는 호스트 이름
REMOTE_HOST="43.201.205.28"

# SSH 키 파일 경로
SSH_KEY_PATH="/benepick.pem"

# 원격 EC2 인스턴스의 사용자 이름
SSH_USER="ubuntu"

# 원격 서버에서 Nginx 설정 파일 수정
ssh -i $SSH_KEY_PATH $SSH_USER@$REMOTE_HOST << EOF
```

```

sudo sed -i "s/${BEFORE_MAIN_PORT}/${AFTER_MAIN_PORT}/" $NGINX_CONFIG_FILE
sudo sed -i "s/${BEFORE_BANK_PORT}/${AFTER_BANK_PORT}/" $NGINX_CONFIG_FILE
sudo nginx -s reload
EOF

echo "Deploy Completed!!"

# 기존 컨테이너 삭제
echo "$BEFORE_COLOR server down(main_port:${BEFORE_MAIN_PORT}, bank_port:${BEFORE_BANK_PORT})"
docker-compose -p benepick-${BEFORE_COLOR} -f /var/jenkins_home/workspace/benepick/docker-compose/docker-compose.${BEFORE_COLOR}.yaml

```

젠킨스 컨테이너 내부에 도커 컴포즈 설치

▼ docker-compose란?

도커 컴포즈는 단일 서버에서 여러개의 컨테이너를 하나의 서비스로 정의해 컨테이너의 묶음으로 관리할 수 있는 작업 환경을 제공하는 관리 도구입니다.

docker-compose를 쓰는 이유?

여러 개의 컨테이너가 하나의 어플리케이션으로 동작할 때 **도커 컴포즈**를 사용하지 않는다면, 이를 테스트하려면 각 컨테이너를 하나씩 생성해야 합니다. 예를 들면, 웹 어플리케이션을 테스트하려면 웹 서버 컨테이너, 데이터베이스 컨테이너 두 개의 컨테이너를 각각 생성해야 합니다.

```

# 젠킨스 컨테이너 접속
docker exec -it jenkins /bin/bash
# 도커 컴포즈 설치
sudo curl -L \
"https://github.com/docker/compose/releases/download/1.28.5/dockercompose-$(uname -s)-$(uname -m)" \
-o /usr/local/bin/docker-compose

```

docker-compose.blue.yml

```

version: '3'
services:
  blue-backend:
    build:
      context: ../backend
    ports:
      - "8081:8081"
    environment: # 환경 변수 추가 부분
      - SMS_KEY=NCSRGCEYXNITRXGO
      - SMS_SECRET_KEY=DJSA1XHJB6HMRZZHSARMBVTZ50SZCEJL
      - SENDER=01099509587

  blue-backend-mydata:
    build:
      context: ../backend-mydata
    ports:
      - "8082:8082"

```

docker-compose.green.yml

```

version: '3'
services:
  green-backend:

```

```

build:
  context: ../backend
ports:
  - "8083:8081"
environment: # 환경 변수 추가 부분
  - SMS_KEY=NCSRGCEYXNITRXGO
  - SMS_SECRET_KEY=DJSA1XHJB6HMRZZHSARMBVTZ50SZCEJL
  - SENDER=01099509587
green-backend-mydata:
  build:
    context: ../backend-mydata
  ports:
    - "8084:8082"

```

ec2 키 젠킨스 컨테이너 내부로 복사

nginx 서버의 포트를 동적으로 변경할 수 있도록 젠킨스에게 권한을 부여하기 위함

```
docker cp ~/.benepick/pem jenkins:/.
```

포팅매뉴얼_Prometheus

```

# 프로메테우스는 9090포트를 기본적으로 사용하므로 포트를 허용해준다
sudo ufw allow 9090

```

1. 프로메테우스 설정 파일 생성

```

# 설정파일을 저장할 디렉터리 생성
sudo mkdir /root/prom
# 설정파일 생성
sudo vi /prom/prometheus.yml

```



/prom/prometheus.yml

```

# 기본적인 전역 설정
global:
  scrape_interval: 15s # 15초마다 메트릭을 수집한다. 기본은 1분이다.
  evaluation_interval: 15s # 15초마다 메트릭을 수집한다. 기본은 1분이다.
  # 'scrpae_timeout' 이라는 설정은 기본적으로 10초로 세팅되어 있다.
# Alertmanager 설정
alerting:
  alertmanagers:
    - static_configs:
      - targets:
          # - alertmanager:9093
# 규칙을 처음 한번 로딩하고 'evaluation_interval' 설정에 따라 정기적으로 규칙을 평가한다.
rule_files:
  # - "first_rules.yml"
  # - "second_rules.yml"
# 메트릭을 수집할 엔드포인트를 설정. 여기서는 Prometheus 서버 자신을 가리키는 설정을 했다.
scrape_configs:
  # 이 설정에서 수집한 타임시리즈에 'job=<job_name>'으로 잡의 이름을 설정한다.
  - job_name: 'prometheus'
    # 'metrics_path'라는 설정의 기본 값은 '/metrics'이고
    # 'scheme'라는 설정의 기본 값은 'http'이다.
    static_configs:
      - targets: ['localhost:9090']

- job_name: 'benepick1_resource'

```



```
static_configs:
  - targets: ['15.164.146.106:9100']

- job_name: 'benepick2_resource'
  static_configs:
    - targets: ['3.36.207.27:9100']

- job_name: 'spring-actuator-prometheus'
  metrics_path: '/actuator/prometheus' # Application prometheus endpoint
  static_configs:
    - targets: ['benepick.shop'] # Application host:port

- job_name: "mysqld-exporter"
  static_configs:
    - targets: ["j9a610.p.ssafy.io:9104"]

- job_name: "redis"
  static_configs:
    - targets: ["j9a610.p.ssafy.io:9121"]
```

2. 프로메테우스 도커 실행

```
sudo docker run \
  --rm -d \
  -p 9090:9090 \
  -v /prom/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
```

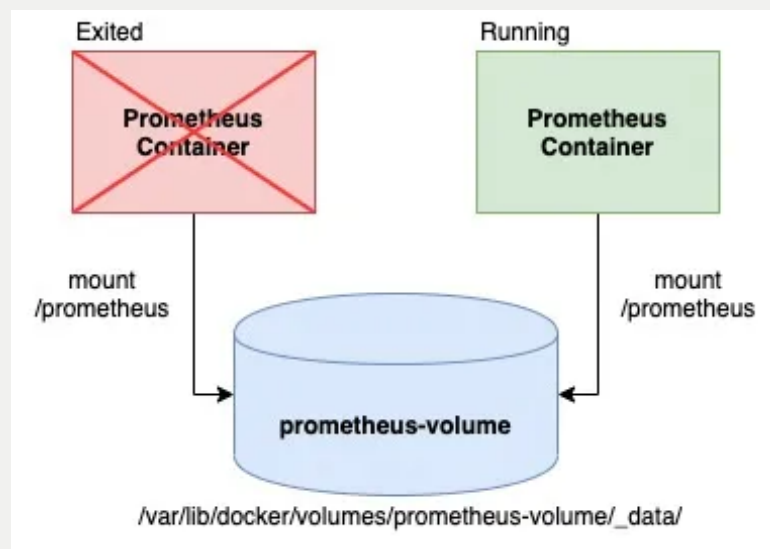
▼ 설명



우리가 위에서 만든 `/prom` 디렉터리 내의 `prometheus.yml` 파일과, 프로메테우스 컨테이너 내부의 `/etc/prometheus/prometheus.yml` 을 동기화 시킨다.

즉, 우리가 작성한 `/prom/prometheus.yml` 이 실제 프로메테우스 컨테이너 내부의 `/etc/prometheus/prometheus.yml` 으로 세팅되어 우리가 지정한 설정 값으로 Prometheus 컨테이너가 실행된다는 뜻이다.

그리고 Prometheus 컨테이너가 종료 되더라도 수집하던 매트릭 데이터는 보존하기 위해 `prometheus-volume`이라는 볼륨을 생성하고 그 볼륨에 Prometheus 컨테이너의 `/prometheus` 디렉터리와 마운트 시킨다.



컨테이너가 종료되더라도 데이터는 외부 볼륨에 저장되므로 삭제되지 않음

이러한 방식을 **사이드카** 방식이라고 한다.

3. 노드 익스포터 설치 및 실행

```
# 압축파일 설치
wget https://github.com/prometheus/node_exporter/releases/download/v1.6.1/node_exporter-1.6.1.linux-amd64.tar.gz
# 압축 해제
tar xzvf node_exporter-1.6.1.linux-amd64.tar.gz
# 노드 익스포터 백그라운드 실행
./node_exporter/node_exporter &
```

MySQL 연동

MySQLD Exporter 바이너리 설치

```
wget https://github.com/prometheus/mysqld_exporter/releases/download/v0.14.0/mysqld_exporter-0.14.0.linux-amd64.tar.gz

tar xzvf mysqld_exporter-0.14.0.linux-amd64.tar.gz
cd mysqld_exporter-0.14.0.linux-amd64
```

MySQL에 권한 추가

```
mysql
# 생성된 계정은 mysqld_exporter의 접속용으로 사용된다.
CREATE USER 'exporter'@'localhost' IDENTIFIED BY 'exporter' WITH MAX_USER_CONNECTIONS 3;
GRANT PROCESS, REPLICATION CLIENT, SELECT ON *.* TO 'exporter'@'localhost';
FLUSH PRIVILEGES;
EXIT;
```

MySQLD Exporter에 설정 추가

```
# MySQL 서버에 추가해준 권한을 MySQLD Exporter 설정 파일에 추가해줍니다.
vi mysqld_exporter.cnf

[client]
user=exporter
password=exporter
```

백그라운드로 MySQLD Exporter 실행

- 백그라운드로 MySQLD Exporter를 실행하기 위해 `mysqld_exporter.service` 서비스 파일을 생성합니다.

```
sudo vi /etc/systemd/system/mysqld_exporter.service

[Unit]
Description=MySQL Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=root
Group=root
Type=simple
Restart=always
ExecStart=/home/mysqld_exporter-0.14.0.linux-amd64/mysqld_exporter \
--config.my-cnf /home/mysqld_exporter-0.14.0.linux-amd64/mysqld_exporter.cnf \
--collect.engine_tokudb_status \
--collect.global_status \
--collect.global_variables \
--collect.info_schema.clientstats \
--collect.info_schema.innodb_metrics \
--collect.info_schema.innodb_tablespace \
```

```
--collect.info_schema.innodb_cmp \
--collect.info_schema.innodb_cmpmem \
--collect.info_schema.processlist \
--collect.info_schema.processlist.min_time=0 \
--collect.info_schema.query_response_time \
--collect.info_schema.replica_host \
--collect.info_schema.tables \
--collect.info_schema.tables.databases='*' \
--collect.info_schema.tablestats \
--collect.info_schema.schemastats \
--collect.info_schema.userstats \
--collect.mysql.user \
--collect.perf_schema.eventsstatements \
--collect.perf_schema.eventsstatements.digest_text_limit=120 \
--collect.perf_schema.eventsstatements.limit=250 \
--collect.perf_schema.eventsstatements.timelimit=86400 \
--collect.perf_schema.eventsstatementssum \
--collect.perf_schema.eventswaits \
--collect.perf_schema.file_events \
--collect.perf_schema.file_instances \
--collect.perf_schema.file_instances.remove_prefix=false \
--collect.perf_schema.indexiowaits \
--collect.perf_schema.memory_events \
--collect.perf_schema.memory_events.remove_prefix=false \
--collect.perf_schema.tableiowaits \
--collect.perf_schema.tablelocks \
--collect.perf_schema.replication_group_members \
--collect.perf_schema.replication_group_member_stats \
--collect.perf_schema.replication_applier_status_by_worker \
--collect.slave_status \
--collect.slave_hosts \
--collect.heartbeat \
--collect.heartbeat.database=true \
--collect.heartbeat.table=true \
--collect.heartbeat.utc
--web.listen-address=0.0.0.0:9104

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

- 백그라운드 서비스 등록

```
systemctl daemon-reload
systemctl start mysqld_exporter
systemctl enable mysqld_exporter
systemctl restart mysqld_exporter
systemctl status mysqld_exporter
```

Prometheus 서버 yml 설정 파일 수정

- Prometheus가 설치된 서버에서 MySQLD Exporter의 metrics HTTP endpoint에 접근할 수 있도록 아래와 같이 `prometheus.yml` 파일에 내용을 추가합니다.

```
cd /prom
vi prometheus.yml

scrape_configs:
- job_name: "mysqld-exporter"
  static_configs:
    - targets: ["j9a610.p.ssafy.io:9104"]
```

- 변경 내용을 적용하기 위해 Prometheus를 재시작해줍니다.

```
docker run \
--rm -d \
-p 9090:9090 \
-v /prom/prometheus.yml:/etc/prometheus/prometheus.yml \
prom/prometheus
```

Redis 연동

Redis-Exporter 설치

```
# 설치 기준 v1.54.0
wget https://github.com/oliver006/redis_exporter/releases/download/v1.54.0/redis_exporter-v1.54.0.linux-amd64.tar.gz

tar -zxvf redis_exporter-v1.54.0.linux-amd64.tar.gz
cd redis_exporter-v1.54.0.linux-amd64.tar.gz
// cp redis_exporter /usr/bin/
```

백그라운드 Redis Exporter 실행

```
sudo vi /etc/systemd/system/redis_exporter.service
# 파일을 생성 후에 아래의 내용을 입력합니다.
-----
[Unit]
Description=Redis Exporter
Wants=network-online.target
After=network-online.target

[Service]
User=root
Group=root
Type=simple
ExecStart=/home/redis_exporter-v1.54.0.linux-amd64/redis_exporter \
    --web.listen-address=0.0.0.0:9121 \

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

```
# 입력이 완료되면 아래의 명령을 순서대로 실행합니다.
systemctl daemon-reload
systemctl enable redis_exporter
systemctl start redis_exporter
```

Prometheus 서버 yaml 설정 파일 수정

- Prometheus가 설치된 서버에서 Redis Exporter의 metrics HTTP endpoint에 접근할 수 있도록 아래와 같이 `prometheus.yml` 파일에 내용을 추가합니다.

```
cd /prom
vi prometheus.yml

scrape_configs:
- job_name: "redis"
  static_configs:
    - targets: ["j9a610.p.ssafy.io:9121"]
```

- 변경 내용을 적용하기 위해 Prometheus를 재시작해줍니다.

```
docker run \
  --rm -d \
  -p 9090:9090 \
  -v /prom/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus
```

포팅매뉴얼_Grafana

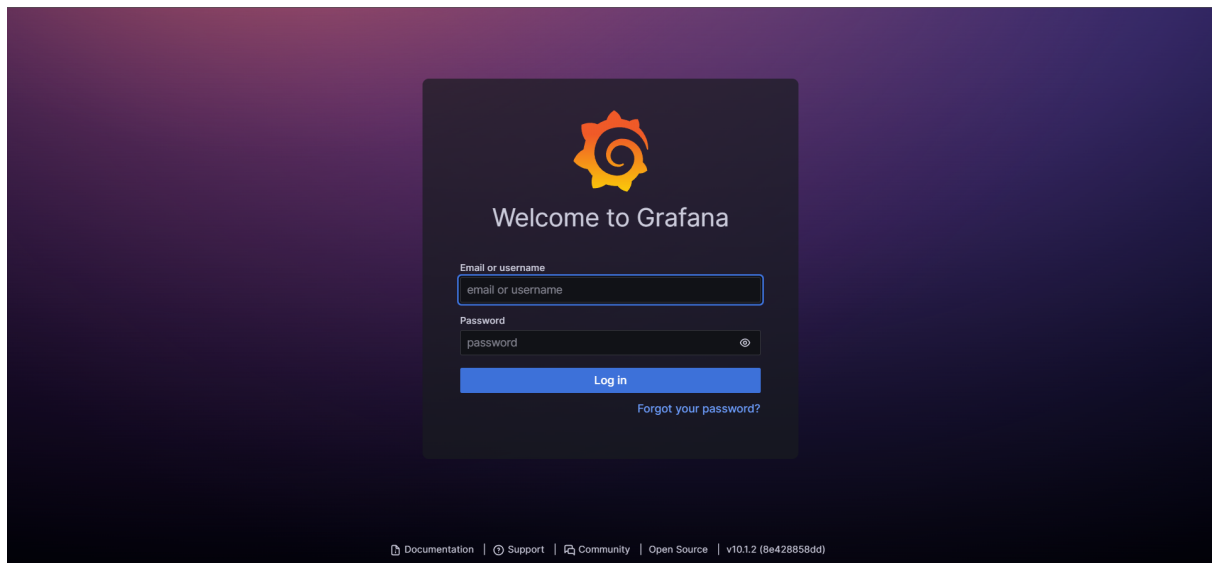
그라파나 설치하기

```
sudo apt-get update

# 그라파나 도커파일 설치
sudo docker pull grafana/grafana:latest

# 필요 폴더 생성 후 권한 설정
sudo mkdir /var/lib/grafana -p
sudo chown -R 472:472 /var/lib/grafana

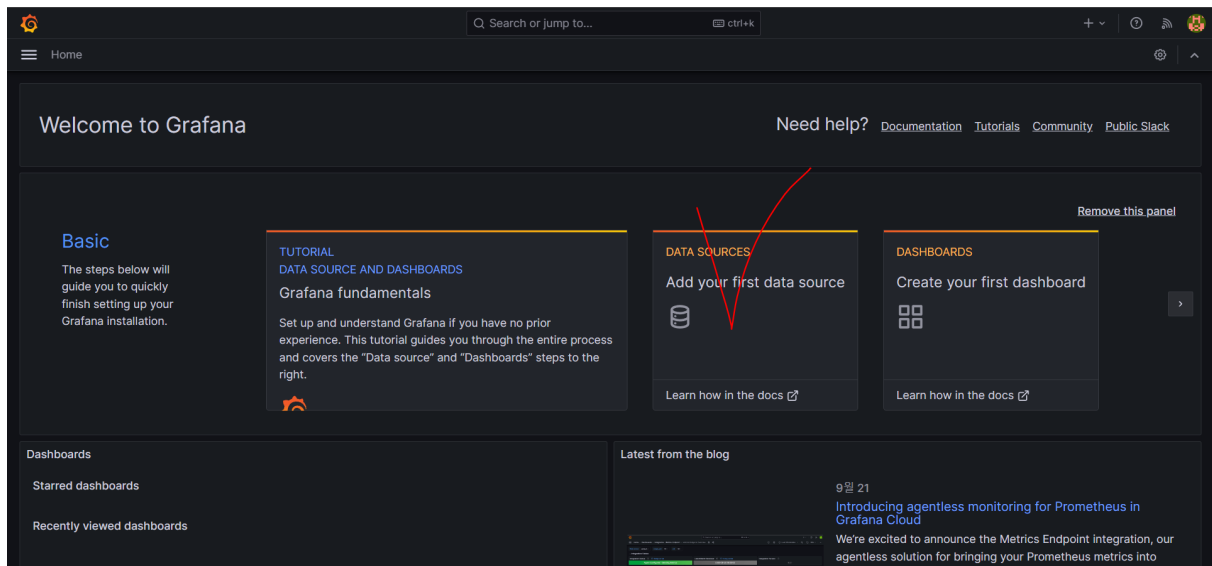
# 그라파나 도커 컨테이너 실행
docker run -d -p 3000:3000 -e "GF_SECURITY_ADMIN_PASSWORD=benepick" grafana/grafana
```



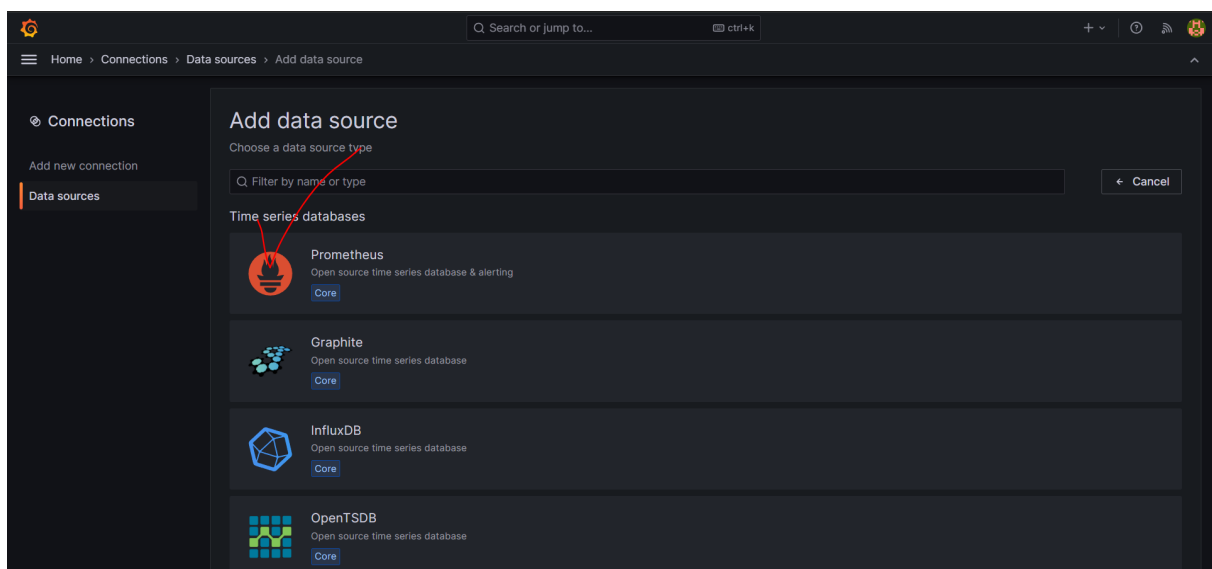
id: admin

password: benepick

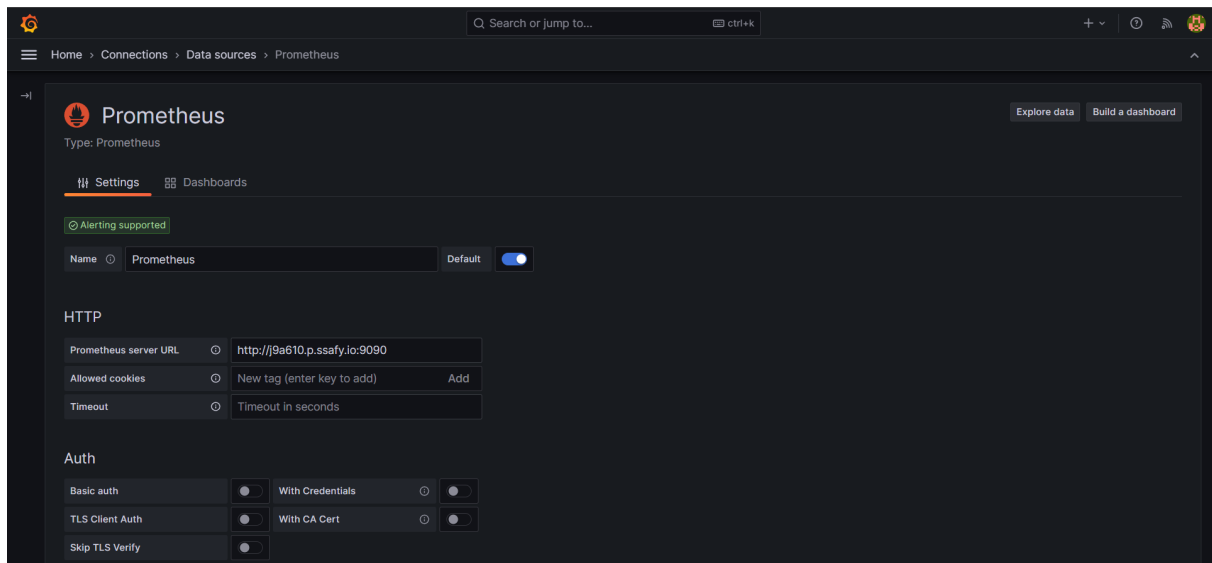
데이터소스 설정 (프로메테우스)



데이터소스 선택

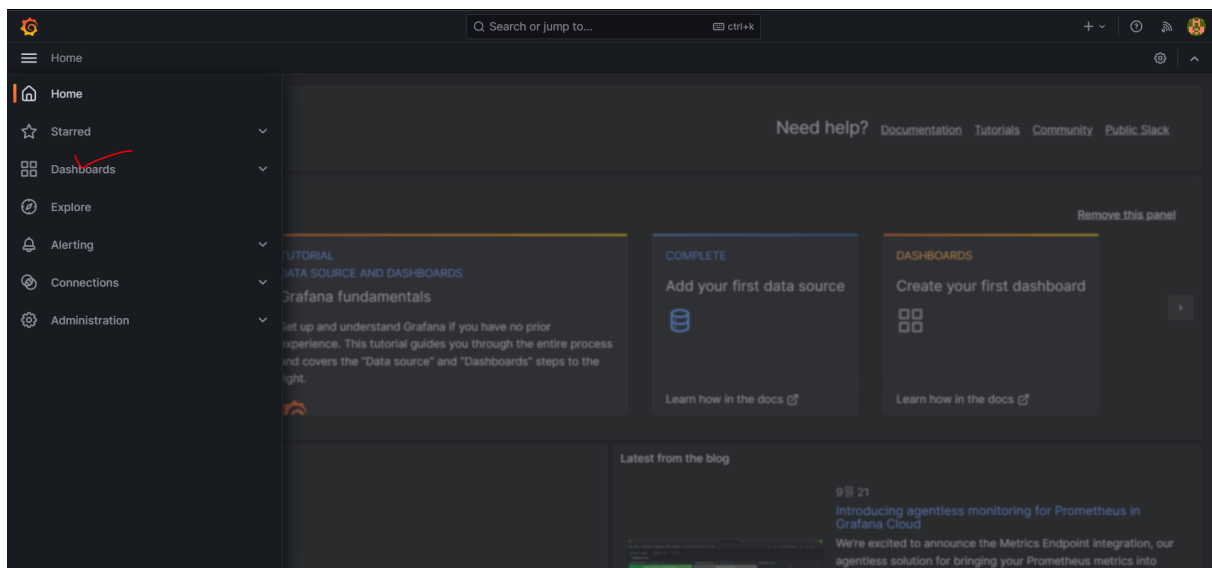


프로메테우스 선택

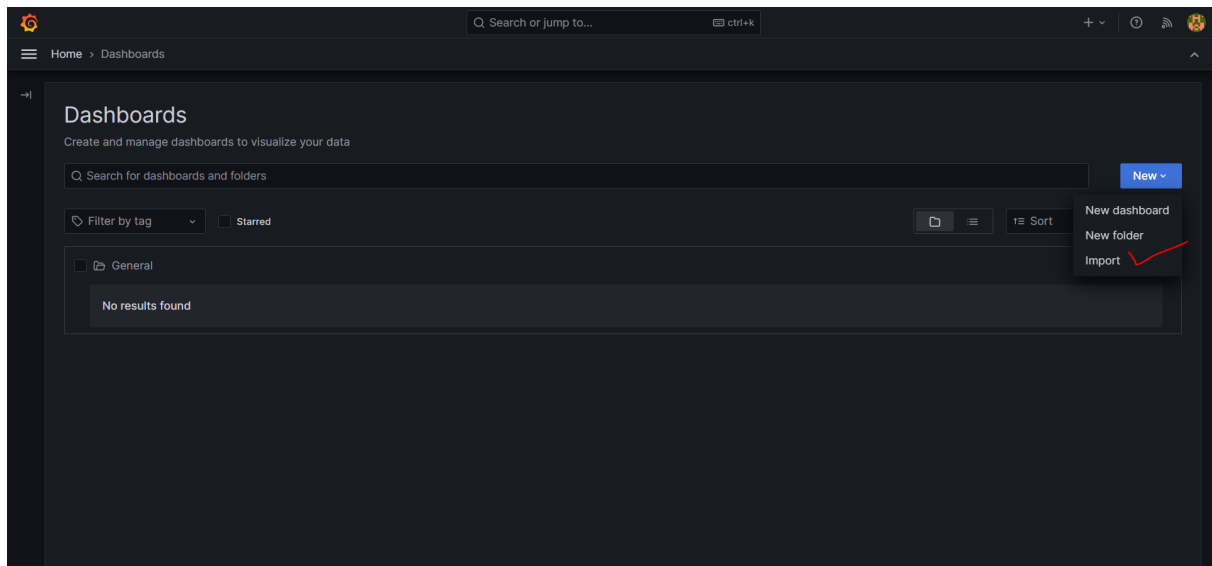


프로메테우스 url 입력 후 나머지 디폴트 설정으로 적용

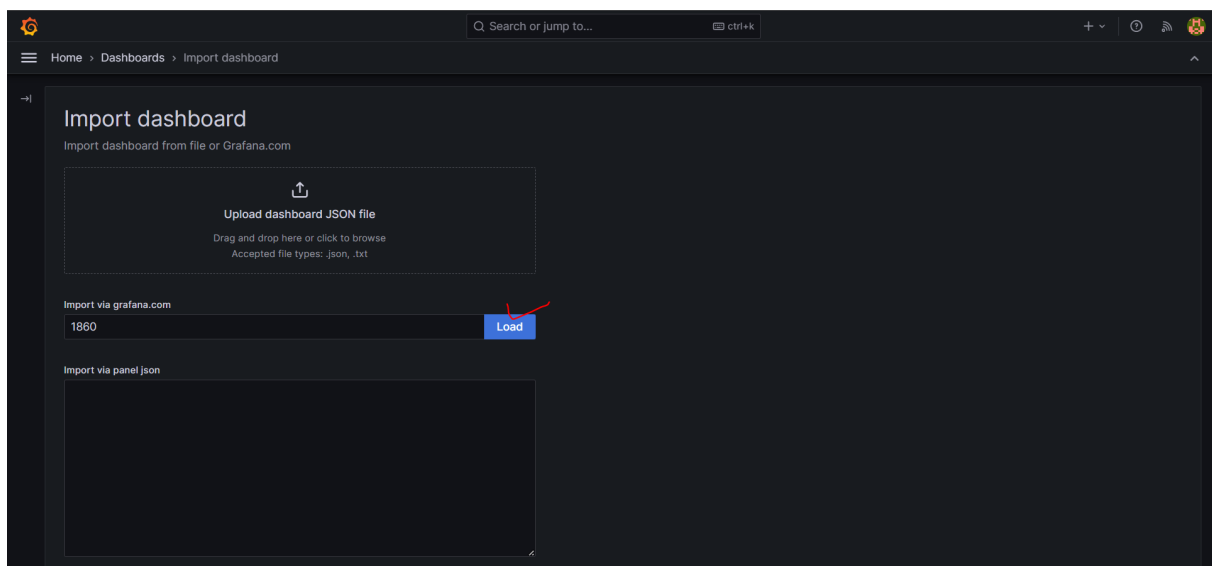
대시보드 템플릿 импорт



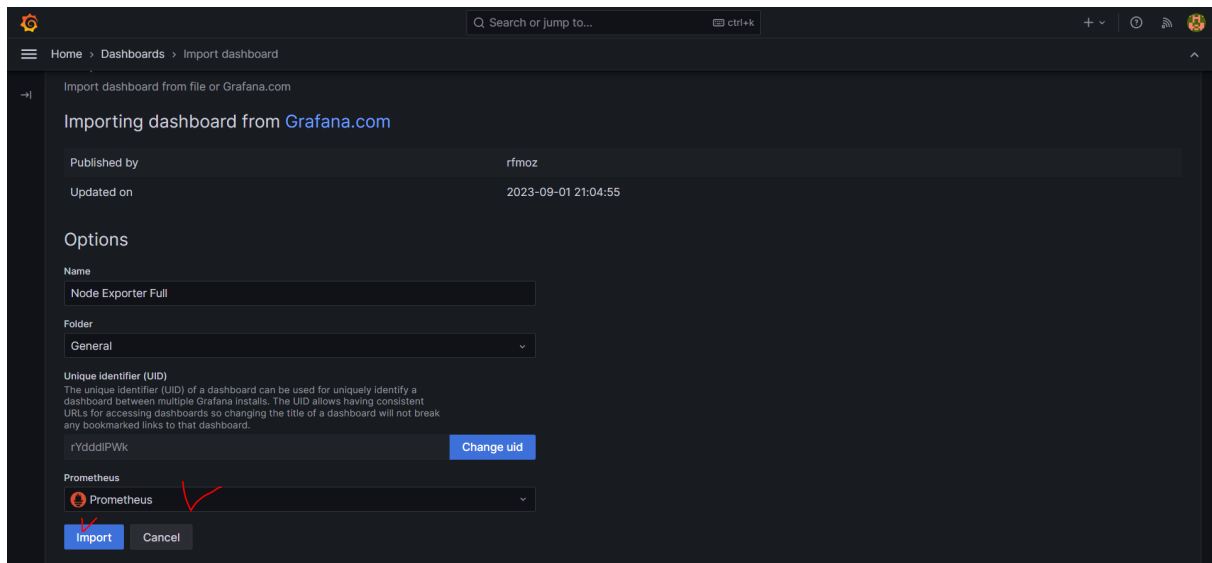
사이드바 열고 대시보드 클릭



New → Import 클릭

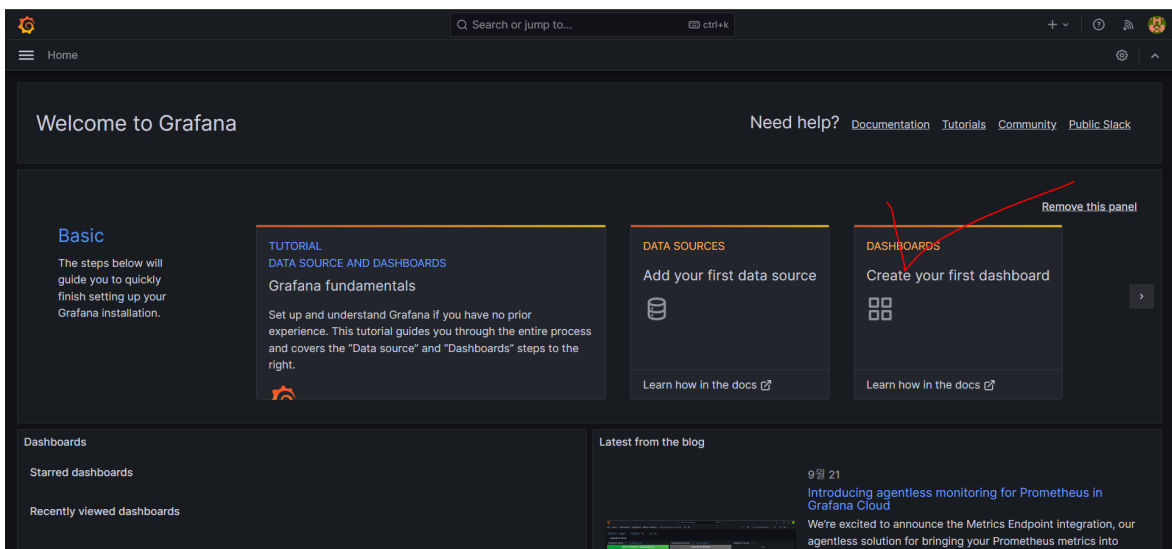


1860(node_exporter), 7362(mysql), 11835(redis)

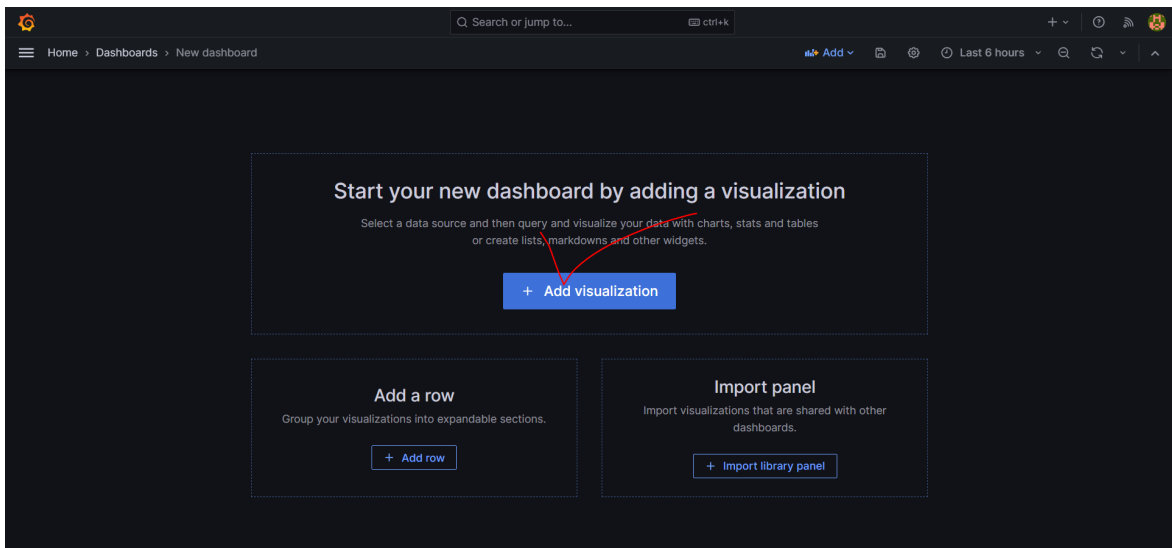


프로메테우스 선택 후, import

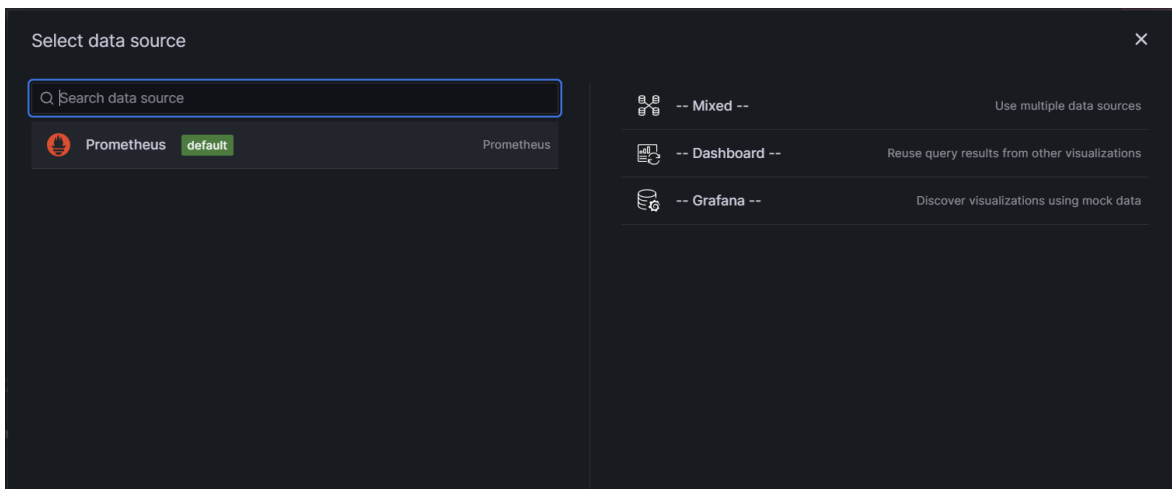
▼ 생성중



Create Dashboard 선택



add vis머시깡이 선택



prometheus 선택

우분투 실행 시 그라파나 및 프로메테우스 자동 실행

```
sudo vi /etc/init.d/autostart.sh
sudo chmod 755 /etc/init.d/autostart.sh
```

```
#!/bin/bash
sudo docker run \
  --rm -d \
  -p 9090:9090 \
  -v /prom/prometheus.yml:/etc/prometheus/prometheus.yml \
  prom/prometheus

sudo docker run -d \
  -p 3000:3000 \
  -e "GF_SECURITY_ADMIN_PASSWORD=benepick" \
  grafana/grafana
```

포팅매뉴얼_테스트서버(nGrinder)

nGrinder Controller 설치

```
# Controller 설치
wget https://github.com/naver/ngrinder/releases/download/ngrinder-3.5.5-20210430/ngrinder-controller-3.5.5.war

# Controller 실행
java -jar ngrinder-controller-3.5.5.war

# 초기 접속
ID : admin
Password : admin
```

nGrinder Agent 설치

- Controller 서버의 ngrinder 접속 후 Agent 다운로드 (ADMIN- 에이전트 관리)

에이전트 관리

All

Keywords

검색

업데이트

정지

추가

다운로드

/agent/download/ngrinder-agent-3.5.5.tar

```
# 다운로드 링크 복사 후 Agent 서버에 위 파일 설치
wget http://...../agent/download/ngrinder-agent-3.5.5.tar

# 다운로드한 tar 파일 압축 해제
sudo tar xvf ngrinder-agent-*.tar

cd ngrinder-agent
```

주의사항

Agent 파일을 UI에서 바로 다운로드하면 **__agent.conf**라는 파일이 함께 생성되는 반면에 위 예시처럼 링크를 통해 다운로드 진행시 해당 파일이 생성되지 않는다. 따라서 직접 작성해줘야 한다.

```
# 해당 폴더에 agent.conf 파일 생성후 아래 내용을 입력한다.
vim agent_conf

-----

common.start_mode=agent
agent.controller_host= Controller IP주소
agent.controller_port=16001
agent.subregion=
agent.owner=
#agent.host_id=
#agent.server_mode=true

# provide more agent java execution option if necessary.
#agent.java_opt=
# provide more agent jvm classpath if necessary.
#agent.jvm.classpath=
# set following false if you want to use more than 1G Xmx memory per a agent process.
#agent.limit_xmx=true
# please uncomment the following option if you want to send all logs to the controller.
```

```
#agent.all_logs=true
# some jvm is not compatible with DNSJava. If so, set this false.
#agent.enable_local_dns=false
# please uncomment the following option if you want to run controller_to_agent connection mode agent.
#agent.connection_mode=controller_to_agent
#agent.connection_port=14000
# set following with the ip you want to broadcast yourself. Set this option if the agent needs to be discovered as public ip.
#agent.broadcast_ip=

-----
# 설정이 완료되면 아래 명령어로 Agent를 실행한다.
./run_agent.sh
```

테스트 스크립트

```
import static net.grinder.script.Grinder.grinder
import org.junit.FixMethodOrder
import org.junit.runners.MethodSorters
import java.util.Random
import java.sql.Connection
import java.sql.DriverManager
import java.sql.ResultSet
import java.sql.Statement
import static org.junit.Assert.*
import static org.hamcrest.Matchers.*
import net.grinder.script.GTest
import net.grinder.script.Grinder
import net.grinder.scriptengine.groovy.junit.GrinderRunner
import net.grinder.scriptengine.groovy.junit.annotation.BeforeProcess
import net.grinder.scriptengine.groovy.junit.annotation.BeforeThread
import org.junit.Before
import org.junit.After
import org.junit.Test
import org.junit.runner.RunWith
import org.ngrinder.http.HTTPRequest
import org.ngrinder.http.HTTPRequestControl
import org.ngrinder.http.HTTPResponse
import groovy.json.JsonBuilder
import groovy.time.TimeCategory

@RunWith(GrinderRunner)
@FixMethodOrder(MethodSorters.NAME_ASCENDING)
class TestRunner {

    public static GTest test01, test02, test03, test04, test05, test06, test07, test08, test09
    public static Map<String, String> headers
    public static HTTPRequest request
    public static List<Map<String, String>> userDataList
    public static List<String> accessTokenList // accessToken을 저장할 전역 변수
    public static Random random
    public static List<Integer> cardIdList

    @BeforeProcess
    public static void beforeProcess() {
        HTTPRequestControl.setConnectionTimeout(300000)
        test01 = new GTest(1, "Place Recommend Test")
        test02 = new GTest(2, "Month Result Test")
        test03 = new GTest(3, "Get Card List And CardInfo")
        test04 = new GTest(4, "Get Card Benefit Test")
        test05 = new GTest(5, "Search Card Benefit in my Card Test")
        test06 = new GTest(6, "Search Card Benefit in all Card Test")
        test07 = new GTest(7, "Get four month Result Test")
        test08 = new GTest(8, "Get Category Result Test")
        test09 = new GTest(9, "Refresh MyData Test")

        random = new Random()
        headers = [:]
        userDataList = []
        cardIdList = []
        accessTokenList = []

        request = new HTTPRequest()
        // Set header data
        headers.put("Content-Type", "application/json")

        // MySQL 데이터베이스에 연결
        def url = "jdbc:mysql://j9a610.p.ssafy.io:3306/benepick_bank"
```

```

def user = "benepick"
def password = "benepick"
// JDBC 드라이버 로드
Class.forName("com.mysql.cj.jdbc.Driver")

Connection con = DriverManager.getConnection(url, user, password)

// 쿼리 실행
Statement stmt = con.createStatement()
ResultSet rs = stmt.executeQuery("SELECT mydata_user_name, mydata_user_social_number, mydata_user_phone_number FROM mydata_user")
int i = 0

// 결과 처리
while (rs.next()) {
    if(i >= 500)
        break

    String userSocialNumberFull = rs.getString("mydata_user_social_number")
    String userSocialNumber = userSocialNumberFull.substring(0, 6)
    String userGenderAndGenerationCode = userSocialNumberFull.substring(7, 8)

    def userData = [
        userName: rs.getString("mydata_user_name"),
        userSocialNumber: userSocialNumber,
        userPhoneNumber: '0' + rs.getString("mydata_user_phone_number"),
        userGenderAndGenerationCode: userGenderAndGenerationCode,
        userSimplePassword: "123456"
    ]
    userDataList.add(userData)
    i++
}

for (int j = 0; j < userDataList.size(); j++) {
    // 코드
    def userData = userDataList[j]
    def jsonBody = new JsonBuilder(userData).toString()
    byte[] payloadBytes = jsonBody.getBytes("UTF-8")
    HTTPResponse response = request.POST("https://benepick.shop/api/user/signup" , payloadBytes)

    if (response.statusCode != 200 && response.statusCode != 470) {
        grinder.logger.warn("Warning. Unexpected response code: ${response.statusCode}")
    } else {
        // "Authorization" 헤더 값 가져오기
        for (header in response.headers) {
            if (header.name == "Authorization" || header.name == "authorization") {
                accessTokenList << header.value
                break
            }
        }
    }
}

// 리소스 해제
rs.close()
stmt.close()
con.close()
}

@BeforeThread
public void beforeThread() {
    test01.record(this, "test01")
    test02.record(this, "test02")
    test03.record(this, "test03")
    test04.record(this, "test04")
    test05.record(this, "test05")
    test06.record(this, "test06")
    test07.record(this, "test07")
    test08.record(this, "test08")
    test09.record(this, "test09")

    request.setHeaders(headers)
    grinder.statistics.delayReports = false
    grinder.logger.info("before thread.")
}

@After // 각 테스트 메서드 실행 후에 호출되는 메서드 -> THINK TIME
public void delayAfterEachTest() {
    try {
        Thread.sleep(5000) // 5초 대기
    } catch (InterruptedException e) {
        e.printStackTrace()
    }
}

```

```

@Test
public void test01() {
    request.setHeaders( ["Authorization": accessTokenList[random.nextInt(accessTokenList.size())]] )
    double randomX = 124.000000 + (131.000000 - 124.000000) * random.nextDouble()
    double randomY = 33.000000 + (39.000000 - 33.000000) * random.nextDouble()

    randomX = Math.round(randomX * 1000000) / 1000000.0
    randomY = Math.round(randomY * 1000000) / 1000000.0
    HTTPResponse response = request.GET("https://benepick.shop/api/card/place?x="+ randomX.toString() + "&y=" + randomY.toString())

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
    }
}

@Test
public void test02() {
    HTTPResponse response = request.GET("https://benepick.shop/api/mydata/card/payment")

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
    }
}

@Test
public void test03() {
    HTTPResponse response = request.GET("https://benepick.shop/api/user/card")

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        def parsedJson = new groovy.json.JsonSlurper().parseText(response.getBodyText())
        int cardId = 0
        parsedJson.data.each { item ->
            cardIdList << item.cardId
            cardId << item.cardId
        }
        assertThat(response.statusCode, is(200))
        if(cardId == 0)
            return

        HTTPResponse response2 = request.GET("https://benepick.shop/api/mydata/card/payment/" + cardId.toString() + "?year=2023&month=8")

        if (response2.statusCode == 301 || response2.statusCode == 302) {
            grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response2.statusCode)
        } else {
            assertThat(response2.statusCode, is(200))
        }
    }
}

@Test
public void test04() {
    if(cardIdList.size() == 0)
        return
    int cardId = cardIdList[random.nextInt(cardIdList.size())]
    HTTPResponse response = request.GET("https://benepick.shop/api/card/benefit/" + cardId.toString())

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        cardIdList = []
        assertThat(response.statusCode, is(200))
    }
}

@Test
public void test05() {
    HTTPResponse response = request.GET("https://benepick.shop/api/card/benefit/user/cu" )

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
    }
}

```

```

@Test
public void test06() {
    HTTPResponse response = request.GET("https://benepick.shop/api/card/benefit/all/cu" )

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
    }
}

@Test
public void test07() {
    HTTPResponse response = request.GET("https://benepick.shop/api/mydata/card/payment/recent")

    if(response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
    }
}

@Test
public void test08() {
    HTTPResponse response = request.GET("https://benepick.shop/api/mydata/card/payment/category" )

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
    }
}

@Test
public void test09() {
    HTTPResponse response = request.GET("https://benepick.shop/api/mydata/renewal" )

    if (response.statusCode == 301 || response.statusCode == 302) {
        grinder.logger.warn("Warning. The response may not be correct. The response code was {}.", response.statusCode)
    } else {
        assertThat(response.statusCode, is(200))
        cardIdList = []
    }
}
}

```

포팅매뉴얼_ChatBot서버

ChatBot서버 도커파일 작성

```

FROM python:3.11 as requirements-stage
WORKDIR /tmp
RUN pip install poetry
COPY ./pyproject.toml ./poetry.lock* /tmp/
RUN poetry export -f requirements.txt --output requirements.txt --without-hashes

FROM python:3.11
WORKDIR /code
COPY --from=requirements-stage /tmp/requirements.txt /code/requirements.txt
RUN pip install --no-cache-dir --upgrade -r /code/requirements.txt
COPY . /code/
# Heroku uses PORT, Azure App Services uses WEBSITES_PORT, Fly.io uses 8080 by default
CMD ["sh", "-c", "uvicorn server.main:app --host 0.0.0.0 --port 3001"]

```

도커 이미지 빌드 및 PUSH , PULL

```

# 만약 수동 배포를 한다면
cd [도커파일위치]
docker build -t psg980331/benepick-gpt .
docker push psg980331/benepick-gpt

```

```
# ec2에서 git pull
docker pull psg980331/benepick-gpt
```

ChatBot서버 실행

```
docker run -d -v /home/benepick-chatgpt/cards:/app/cards -e PINECONE_API_KEY=[API_KEY] -e PINECONE_ENVIRONMENT=gcp-starter -e PINECONE
```

포팅매뉴얼_Vector DB 구축

1. Pinecone Vector DB 구축
2. <https://app.pinecone.io/> 서비스 회원가입
3. 전처리 카드 데이터 파일 filtered_benefit.csv 생성
4. 챗봇 서버 실행
5. S09P22A610/chatbot/retrieval plugin/vector db 구축_메타데이터.ipynb 실행

```
# 환경변수 설정
os.environ['PINECONE_API_KEY'] = "MY_PINECONE_API_KEY"
os.environ['PINECONE_ENVIRONMENT'] = "MY_PINECONE_ENVIRONMENT"
os.environ["OPENAI_API_KEY"] = "MY_OPENAI_API_KEY"
os.environ['PINECONE_INDEX'] = "benepick"
os.environ['DATASTORE'] = "pinecone"
```

6. C:\chatgpt-retrieval-plugin\cards 폴더 데이터를 EC2 서버 /home/benepick-chatgpt/cards 경로로 이동

포팅매뉴얼_DB서버

MYSQL 설치

```
# apt 업데이트
sudo apt-get update
# mysql 설치
sudo apt-get install mysql-server
# mysql 포트 허용
sudo ufw allow mysql
# mysql 실행
sudo systemctl start mysql

# mysql 외부 접속 허용
sudo vi /etc/mysql/mysql.conf.d/mysqld.cnf
```

```
#
# Instead of skip-networking the default is now to listen only on
# localhost which is more compatible and is not less secure.
# bind-address            = 127.0.0.1
mysqlx-bind-address       = 127.0.0.1
#
# * Fine Tuning
#
```



```
# 위과 같이 bind-address 주석 처리

# mysql 재실행
sudo systemctl restart mysql

# benepick 유저 생성
create user 'benepick'@'%'identified by 'benepick';
grant all privileges on *.*to 'benepick'@'%';
flush privileges;
```

mysql workbench 접속

Setup New Connection

Connection Name: Type a name for the connection

Connection Method: Method to use to connect to the RDBMS

Parameters SSL Advanced

Hostname: Port: Name or IP address of the server host - and TCP/IP port.

Username: Name of the user to connect with.

Password: The user's password. Will be requested later if it's not set.

Default Schema: The schema to use as default schema. Leave blank to select it later.

Redis 설치

```
#Redis 서버 설치 및 버전 확인
sudo apt-get update
sudo apt-get install redis-server
redis-server --version

#Redis 설정 파일 수정
sudo vi /etc/redis/redis.conf

#원격 액세스 가 가능하도록 서버를 열어줌
bind 0.0.0.0 ::1

#메모리 최대 사용 용량 및 메모리 초과시 오래된 데이터를 지워 메모리 확보하도록 정책 설정
maxmemory 2g
maxmemory-policy allkeys-lru

sudo systemctl restart redis-server

#Redis 설정 확인
sudo systemctl status redis-server
redis-cli ping

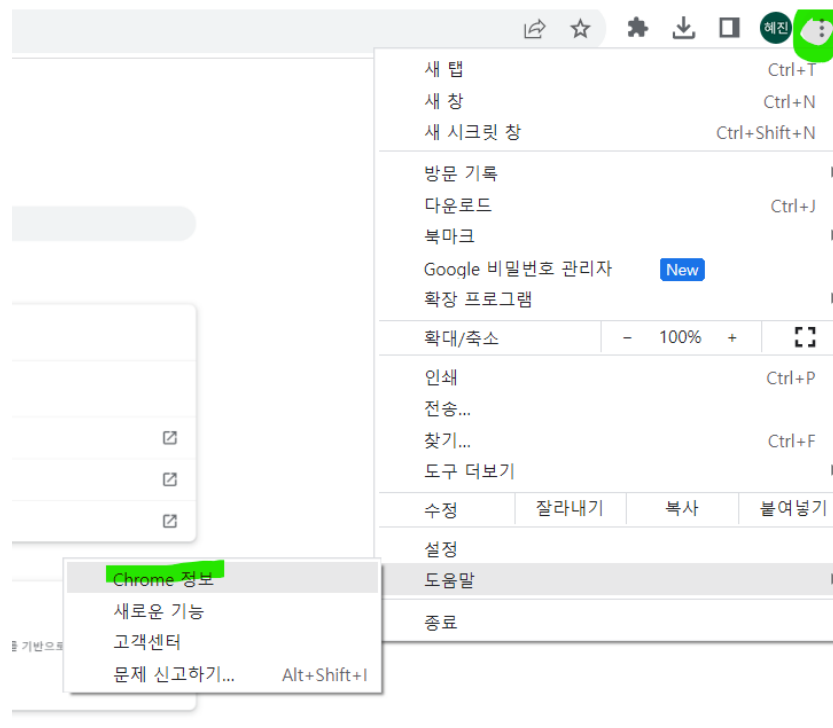
#명령어를 이용해 Redis 서버 접속 가능
redis-cli
```

포팅매뉴얼_ChromeDriver

1. Chrome Driver 설치하기

Chrome 버전 확인하기

- 1) 크롬 실행 후
- 2) 브라우저 오른쪽 상단의 점 세개 클릭
- 3) 도움말 -> Chrome 정보를 선택



설정

- 나와 Google의 관계
 - 자동 완성 및 비밀번호
 - 개인 정보 보호 및 보안
 - 성능
 - 모양
 - 검색엔진
 - 기본 브라우저
 - 시작 그룹
-
- 언어
 - 다운로드
 - 접근성
 - 시스템
 - 설정 초기화

확장 프로그램

Chrome 정보

설정 검색

Chrome 정보

Google Chrome

Chrome 업데이트 중(69%)
버전 119.0.5993.132(공식 빌드) (64비트)

Chrome 도움말 보기



문제 신고



개인정보처리방침



Google Chrome
Copyright 2023 Google LLC. All rights reserved.

Chrome은 [Chromium](#) 오픈소스 프로젝트를 비롯한 여러 [오픈소스 소프트웨어](#)를 기반으로 제작된 브라우저입니다

[서비스 약관](#)

Chrome Driver 설치하기

구글에 Chromedriver를 검색하거나, 아래의 Chromedriver 다운로드 페이지로 이동합니다.

Chromedriver - WebDriver for Chrome - Downloads

Current Releases If you are using Chrome version 115 or newer, please consult the Chrome for Testing availability dashboard. This page provides convenient JSON endpoints for specific Chromedriver version downloading. For older versions of Chrome, please see below for the version of Chromedriver

<https://chromedriver.chromium.org/downloads>

1) 버전에 맞는 Chromedriver를 선택한 후, 다운로드를 진행합니다.

Current Releases

- If you are using Chrome version 115 or newer, please consult [the Chrome for Testing availability dashboard](#). This page provides convenient JSON endpoints for specific ChromeDriver version downloading.
- For older versions of Chrome, please see below for the version of ChromeDriver that supports it.

For more information on selecting the right version of ChromeDriver, please see the [Version Selection](#) page.

ChromeDriver 114.0.5735.90

Supports Chrome version 114

For more details, please see the [release notes](#).

ChromeDriver 114.0.5735.16

Supports Chrome version 114

For more details, please see the [release notes](#).

ChromeDriver 113.0.5672.63

Supports Chrome version 113

- Resolved issue 4205: Same object ids in Classic and BiDi [Pri-1]
- Resolved issue 4302: Don't assume that Mapper is in the first tab in ExecuteGetWindowHandles [Pri-1]
- Resolved issue 4356: Chrome 110 not utilizing pref value "download.default_directory" [Pri-1]

For more details, please see the [release notes](#).

ChromeDriver 113.0.5672.24

Supports Chrome version 113

🔍

2) 압축 풀기를 진행한 후, 별도의 설치 없이 사용할 곳에 이동시켜 사용하면 됩니다.

chrome-win64					
공유 보기					
📁 > 내 PC > 바탕 화면 > crawling > data_crawling > chrome-win64					
기 본 화 면	이름	수정된 날짜	유형	크기	
소개서 ive 체 본	📁 chrome-win64	2023-08-31 오후 2:48	파일 폴더		
	📁 data	2023-09-04 오전 9:16	파일 폴더		
	📄 crawling.ipynb	2023-08-31 오후 3:23	IPYNB 파일	17KB	
	📄 hh.ipynb	2023-08-31 오후 4:04	IPYNB 파일	6KB	
	📄 new.ipynb	2023-08-31 오후 4:04	IPYNB 파일	8KB	
	📄 nn.ipynb	2023-08-31 오후 5:50	IPYNB 파일	14KB	
	📄 nn2.ipynb	2023-09-01 오후 5:40	IPYNB 파일	4,600KB	
	📄 working.ipynb	2023-08-31 오후 2:34	IPYNB 파일	16KB	

3) data/1_crawling/crawling.ipynb을 실행하면 데이터를 크롤링이 실행됩니다.

포팅매뉴얼_프론트

1. Node LTS(v18.17.1)설치

2. chocolately 설치(<https://chocolatey.org/>)

- 관리자 권한으로 cmd 창 실행 후에

```
choco install -y nodejs-lts microsoft-openjdk11
```

3. Android SDK 2022.3.1.19(버전 크게 상관없음)

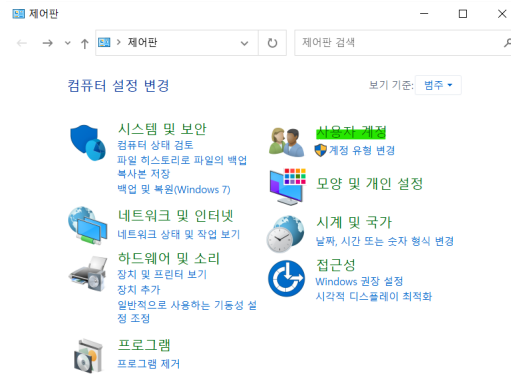
- ## 설치 전 체크

설치 후 환경변수 설정 필수

1. ANDROID_HOME 환경 변수 구성

React Native 도구에서는 네이티브 코드로 앱을 빌드하기 위해 일부 환경 변수를 설정해야 합니다.

1. Windows 제어판 - 사용자 계정



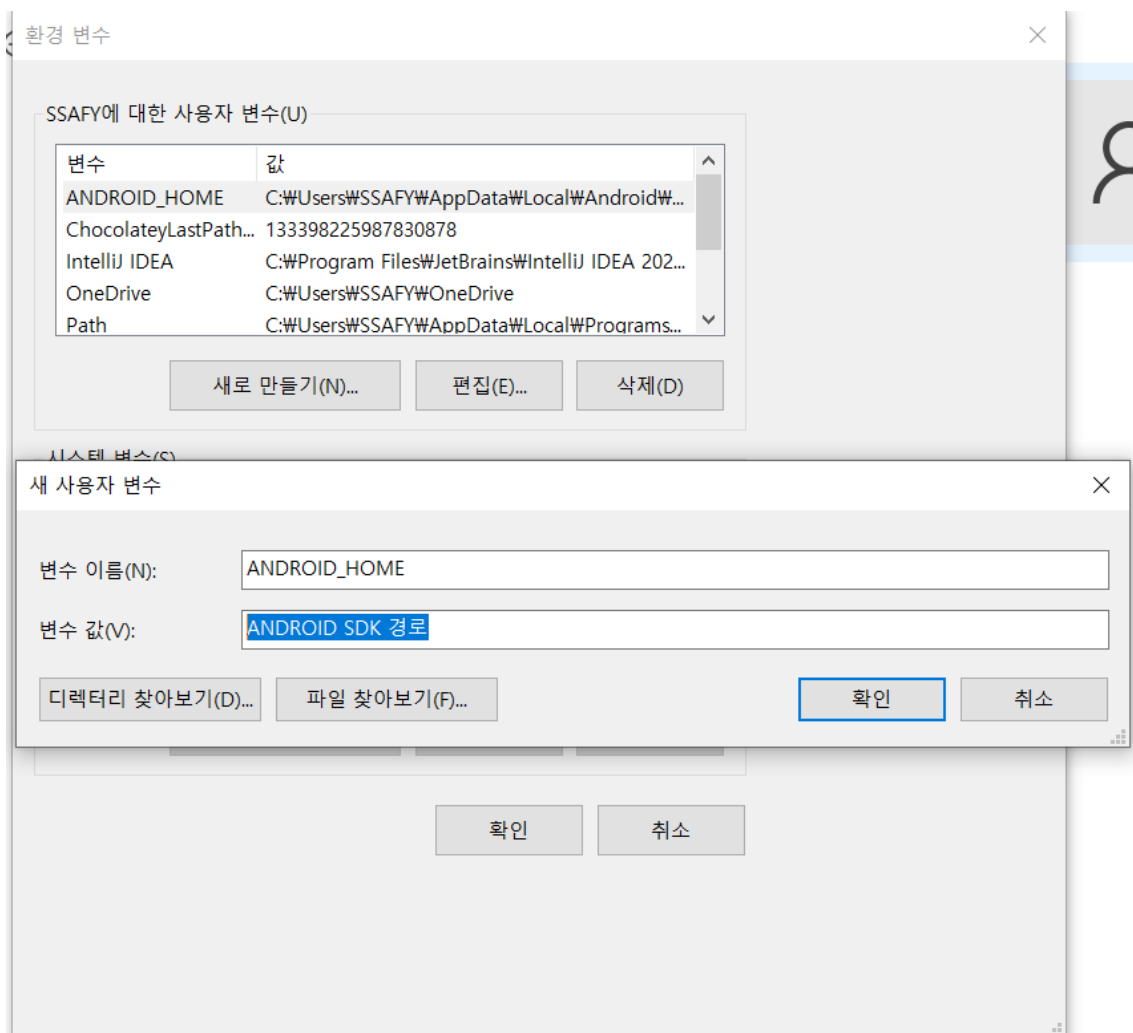
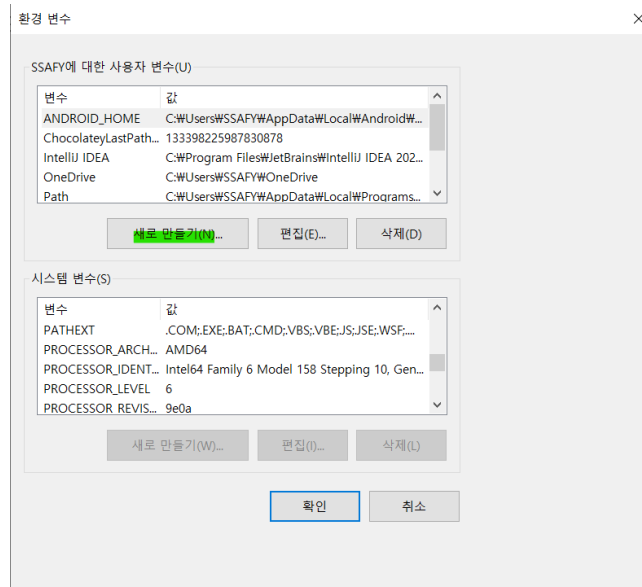
2. 사용자 계정 - 환경 변수 변경

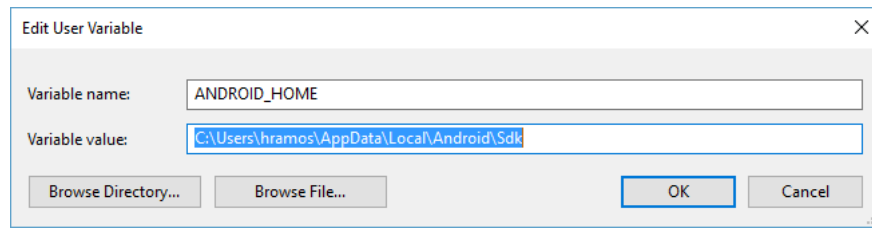


3. 새로 만들기...를 `ANDROID_HOME`

클릭하여

Android SDK 경로를 가리키는 새 사용자 변수를 만듭니다 .





SDK는 기본적으로 다음 위치에 설치됩니다.

%LOCALAPPDATA%\Android\Sdk

Android Studio '설정' 대화상자의 **Appearance & Behavior** → **System Settings** → **Android SDK** 에서 SDK의 실제 위치를 찾을 수 있습니다 .

다음 단계를 진행하기 전에 새 명령 프롬프트 창을 열어 새 환경 변수가 로드되었는지 확인하세요.

1. 파워셸 열기

2. Get-ChildItem -Path Env:

를 복사하여 powershell에 붙여 넣습니다.

3. **ANDROID_HOME** 추가 확인

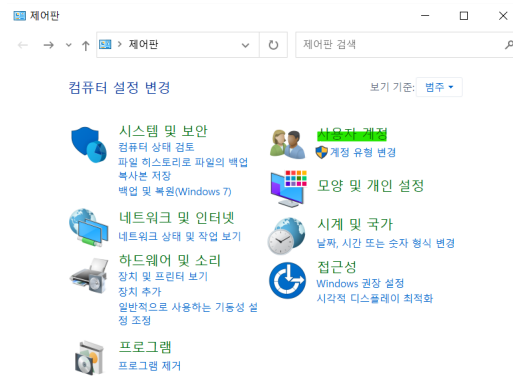
관리자: Windows PowerShell

```
PS C:\windows\system32> Get-ChildItem -Path Env:

Name                           Value
----                           -
ALLUSERSPROFILE                C:\ProgramData
ANDROID_HOME                  C:\Users\SSAFY\AppData\Local\Android\Sdk
APPDATA                        C:\Users\SSAFY\AppData\Roaming
ChocolateyInstall              C:\ProgramData\chocolatey
ChocolateyLastPathUpdate       133398225987830878
CommonProgramFiles             C:\Program Files\Common Files
CommonProgramFiles(x86)        C:\Program Files (x86)\Common Files
CommonProgramW6432             C:\Program Files\Common Files
COMPUTERNAME                   DESKTOP-BLHP263
ComSpec                        C:\windows\system32\cmd.exe
DriverData                     C:\Windows\System32\Drivers\DriverData
GIT_LFS_PATH                   C:\Program Files\Git LFS
HOMEDRIVE                      C:
HOMEPATH                      \Users\SSAFY
IntelliJ IDEA                  C:\Program Files\JetBrains\IntelliJ IDEA 2023.1.4\bin;
LOCALAPPDATA                   C:\Users\SSAFY\AppData\Local
LOGONSERVER                     \\DESKTOP-BLHP263
NUMBER_OF_PROCESSORS           12
```

2. Path에 플랫폼 도구 추가

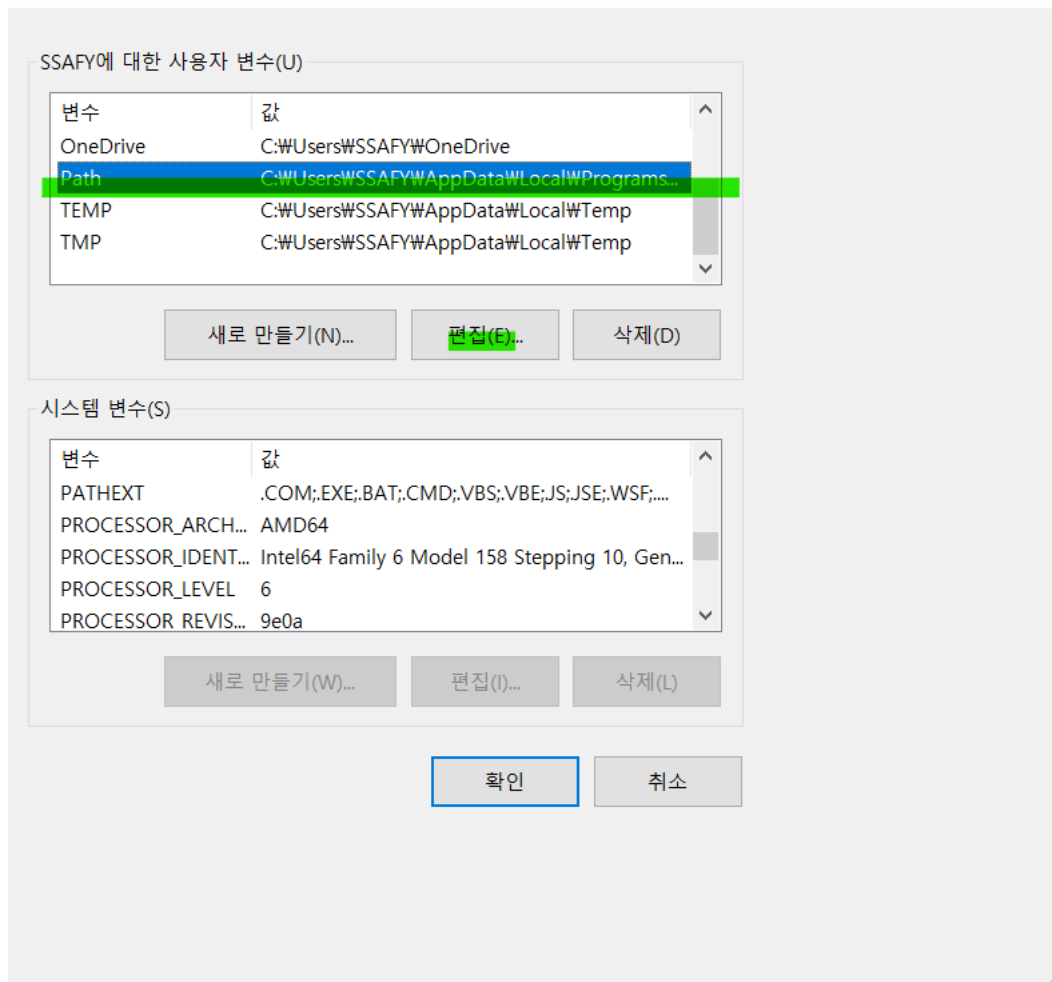
1. Windows 제어판을 엽니다 .
2. 사용자 계정을 클릭하세요.



2. 내 환경 변수 변경을 클릭하세요.



3. 'PATH' - 편집

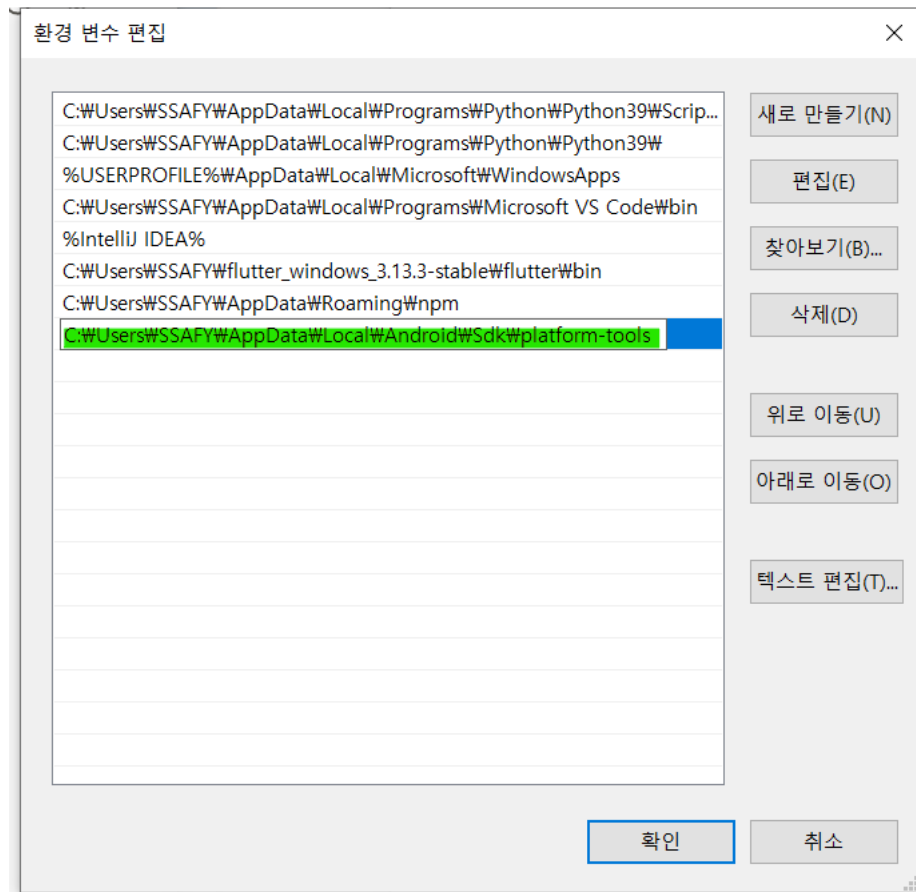


4. 새로 만들기를

클릭 하고 목록에 플랫폼 도구 경로를 추가합니다.

이 폴더의 기본 위치는 다음과 같습니다.

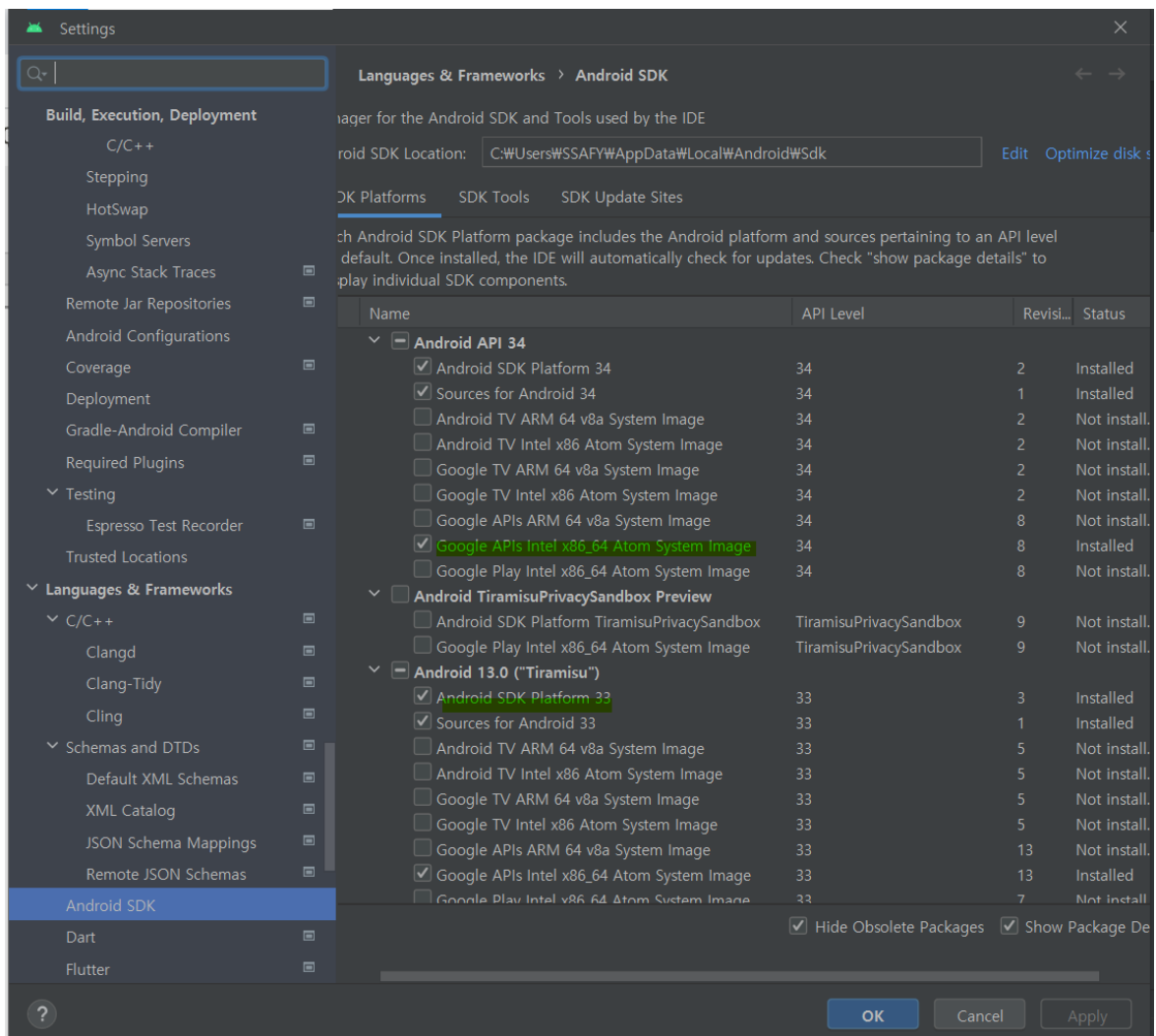
```
%LOCALAPPDATA%\Android\Sdk\platform-tools
```



<https://reactnative.dev/docs/environment-setup>

4. Android SDK 설치 후

- settings 들어간 후에 Android SDK 검색
- SDK platforms 탭에 들어간후에
 - Android SDK Platform 33
 - Google APIs Intel x86_64 Atom System Image
 - 확장 팩 설치



5. 프로젝트 시작

```
npm install
npm start
```