포팅 메뉴얼

1. 빌드 및 배포 정리

우분투 서버 기본 세팅

- ec2 서버의 시간을 한국 표준시로 변경(UTC +0 → UTC +9)

```
sudo timedatectl set-timezone Asia/Seoul
```

- 미러 서버를 카카오 서버로 변경

  why? 해외 서버를 사용하게 되면 패키지 update/upgrade 속도가 매우 느리기 때문

```
sudo sed -i 's/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/g' /etc/apt/sour
ces.list
```

- 패키지 목록 업데이트 및 패키지 업데이트

  ```
  sudo apt update && sudo apt upgrade
  ```

Docker

도커 설치전 필요한 패키지 설치

```
sudo apt install apt-transport-https ca-certificates curl gnupg-agent software-propert
ies-common
```

1. apt-transport-https : HTTPS를 통해 ‘apt’ 패키지를 다운로드 가능하게 해준다. for 보안 강화
2. ca-certificates : 일반적으로 TLS/SSL을 통한 보안 통신을 할때 필요한 CA(Certificatte Authority)인증서를 시스템에 설치. 이 인증서들은 HTTPS 통신 시 신뢰할 수 있는 CA 로부터 발급받은 인증서인지 검증하는데 사용
3. curl : 웹에서 데이터를 전송받는 유틸리티. HTTP, HTTPS 등 다양한 프로토콜을 지원합니다. 이를 통해 예를 들어 도커의 GPG 키를 다운로드할 수 있다.

4. gnupg-agent : GPG(GNU Privacy Guard) 키 관리를 위한 에이전트입니다. 이는 GPG 키의 안전한 보관과 사용을 도와줍니다
5. software-properties-common: PPA를 추가하거나 제거할 때 사용한다.
    a. PPA? Personal Package Archive(개인 패키지 저장소)를 의미하며, 캐노니컬 사의 우분투에서 기본적으로 제공하는 패키지 외의 사적으로 만든 패키지를 의미한다.

도커에 대한 GPG key 인증 진행

```
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
```

- gpg키는 도커에만 존재하는 것은 아니고, 리눅스 패키지 관리 툴(apt)이 이 프로그램 패키지가 유효한지 확인하기 위해 설치 전 gpg키로 검증하는 과정을 거친다. 그래서 curl로 gpg키를 다운받아 apt 키 리스트에 추가해야 한다.

Docker Repository 등록

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

- sudo add-apt-repository 는 Ubuntu계열 리눅스에서 외부 APT Repository를 시스템에 추가하는데 사용한다. 이 명령어를 통해 외부에서 제공하는 패키지를 APT 패키지 관리자로 설치하거나 업데이트 할 수 있다.

    ▼ 명령어 상세 설명

    - **deb**: 저장소의 형식이 DEB 패키지를 사용하는 것을 나타낸다.
    - **[arch=amd64]**: 아키텍처가 AMD64(일반적으로 x86_64라고도 함)인 경우에만 이 저장소를 사용하겠다는 의미입니다.
    - **https://download.docker.com/linux/ubuntu**: 도커의 공식 Ubuntu 리포지토리 URL.
    - **$(lsb_release -cs)**: 현재 시스템의 Ubuntu 버전을 자동으로 감지하여 입력한다. 예를 들어, 시스템이 Ubuntu 20.04(Focal Fossa)를 실행 중이라면, 이 부분은 자동으로 `focal`로 대체됩니다.
    - **stable**: 도커의 안정된(stable) 버전을 설치하겠다는 의미입니다. 다른 옵션으로는 `test`나 `nightly` 등이 있을 수 있습니다.

패키지 리스트 갱신

```
sudo apt update
```

Docker 패키지 설치

```
sudo apt install docker-ce docker-ce-cli containerd.io
```

- apt 패키지 관리자를 이용하여 Docker를 설치한다.→ docker 명령어를 통해 컨테이너를 생성,실행,관리할 수 있게 된다.

▼ 명령어 상세 설명

docker-ce :Docker community edition 설치

docker-ce-cli: Docker CE 에 대한 CLI를 설치. 이를 통해 docker 명령어를 사용할 수 있게 된다.

containerd.io:`containerrd` 컨테이너의 생명주기를 관리하는데 필요한 모든 기능을 제공

Docker 일반 유저에게 권한 부여

```
sudo usermod -aG docker ubuntu
```

- Docker는 항상 root로 실행되기 때문에 sudo 를 사용하여 명령어를 입력해야 한다
- 사용자를 docker 그룹에 추가하여 sudo를 사용하지 않아도 docker 명령어를 사용한다.

Docker 서비스 재시작 및 재로그인

```
sudo service docker restart
exit
```

Docker에서 실행되고 있는 컨테이너 로그 조회

```
docker logs {컨테이너명}
```

- but, 조회되는 시간은 UTC-0

- 오류 확인시 log 확인을 위한 기능

Jenkins (Docker)

-Jenkins를 사용하는 이유?

자동화

- **빌드 자동화**: 소스 코드의 변경을 감지하고 자동으로 빌드를 수행
- **테스트 자동화**: 코드 변경 후 자동으로 테스트를 실행하여 빠르게 오류 탐지
- **배포 자동화**: 코드가 모든 테스트를 통과하면 자동으로 프로덕션 환경으로 배포 가능

유연성과 확장성

- **플러그인 아키텍처**: 다양한 플러그인을 통해 기능을 확장
- **다양한 환경 지원**: 다양한 빌드 및 배포 환경을 지원

협업 촉진

- **버전 관리와의 통합**: Git, SVN과 같은 버전 관리 시스템과 쉽게 통합
- **팀 협업**: 여러 개발자가 동시에 작업을 할 때 코드 통합 문제를 미리 발견하고 해결

모니터링
오픈소스 프로젝트

Jenkins 설치(Docker)

Jenkins 이미지 받기

- docker의 pull 명령을 통해 Lts 버전의 jenkins 이미지를 다운로드 한다.

```
docker pull jenkins/jenkins:lts
```

Jenkins 컨테이너 실행

```
docker run -d \
--env JENKINS_OPTS=--httpPort=8080 \
-v /etc/localtime:/etc/localtime:ro \
```

```
-e TZ=Asia/Seoul \
-p 8080:8080 \
-v /jenkins:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-v /usr/local/bin/docker-compose:/usr/local/bin/docker-compose \
--name jenkins \
-u root \
jenkins/jenkins:lts
```

▼ 명령어 상세 설명

- `d`: 컨테이너를 백그라운드에서 실행.
- `-env JENKINS_OPTS=--httpPort=8080`: Jenkins를 8080 포트에서 실행하도록 환경 변수를 설정합니다.
- `v /etc/localtime:/etc/localtime:ro`: 호스트의 시간 정보를 컨테이너와 동기화. `ro`는 read-only를 의미.
- `e TZ=Asia/Seoul`: 컨테이너의 타임존을 서울로 설정.
- `p 8080:8080`: 호스트의 8080 포트와 컨테이너의 8080 포트를 매핑.
- `v /jenkins:/var/jenkins_home`: 호스트의 `/jenkins` 디렉토리와 컨테이너의 `/var/jenkins_home` 디렉토리를 볼륨으로 연결.
- `v /var/run/docker.sock:/var/run/docker.sock`: Docker 소켓을 컨테이너 내부와 연결하여 컨테이너 내부에서도 Docker 명령을 사용할 수 있게 함.
- `v /usr/local/bin/docker-compose:/usr/local/bin/docker-compose`: 호스트의 `docker-compose` 바이너리를 컨테이너에 공유.
- `-name jenkins`: 컨테이너의 이름을 "jenkins"로 설정합니다.
- `u root`: 컨테이너를 root 사용자로 실행.
- `jenkins/jenkins:lts`: 사용할 이미지를 지정합니다. 여기서는 Jenkins의 LTS(Long Term Support) 버전을 사용.

방화벽 개방

```
sudo ufw allow 8080
```

- ec2 서버의 방화벽 설정을 통해 포트를 개방해 외부 접속이 가능하게 한다.

Jenkins 접속

```
docker exec -it jenkins /bin/bash
```

- jenkins 컨테이너에 접속하기 위해 bash 쉘을 이용한다.

```
netstat -nltp
```

```
ubuntu@ip-172-26-6-113:~$ netstat -nltp
(Not all processes could be identified, non-owned process info
 will not be shown, you would have to be root to see it all.)
Active Internet connections (only servers)
Proto Recv-Q Send-Q Local Address           Foreign Address         State       PID/Program name
tcp        0      0 0.0.0.0:3306            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:6379            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8080            0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:80              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:8081            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:45779         0.0.0.0:*               LISTEN      4954/code-8b617bd08
tcp        0      0 127.0.0.53:53           0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:37749         0.0.0.0:*               LISTEN      5417/code-8b617bd08
tcp        0      0 0.0.0.0:22              0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:3000            0.0.0.0:*               LISTEN      -
tcp        0      0 127.0.0.1:6011          0.0.0.0:*               LISTEN      -
tcp        0      0 0.0.0.0:443             0.0.0.0:*               LISTEN      -
tcp6       0      0 :::8080                 :::*                    LISTEN      -
tcp6       0      0 :::80                   :::*                    LISTEN      -
tcp6       0      0 :::8081                 :::*                    LISTEN      -
tcp6       0      0 :::22                   :::*                    LISTEN      -
tcp6       0      0 :::3000                 :::*                    LISTEN      -
tcp6       0      0 ::1:6011                :::*                    LISTEN      -
```

sudo systemctl enable docker-jenkins.service

정상적으로 컨테이너가 실행되었는지 확인하기 위해, 상단 명령어를 활용하여 포트가 개방되어 있는지 확인한다

Jenkins OS 재시작 후 자동실행 설정

```
sudo vi /etc/systemd/system/docker-jenkins.service
```

```
[Unit]
Description=docker-jenkins
Wants=docker.service
After=docker.service

[Service]
RemainAfterExit=yes
ExecStart=/usr/bin/docker start jenkins
ExecStop=/usr/bin/docker stop jenkins
```

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

▼ 명령어 상세 설명

`/etc/systemd/system/docker-jenkins.service` 파일에 위와 같은 내용을 넣으면 Docker 컨테이너를 실행하기 위한 systemd 서비스를 설정할 수 있다. 각 섹션과 필드에 대한 설명은 다음과 같다.

- `[Unit]`: 서비스의 메타데이터와 의존성을 정의한다.
  - `Description`: 서비스에 대한 설명을 제공한다.
  - `Wants`: `docker.service`가 있으면 좋지만, 없어도 되는 의존성을 나타낸다.
  - `After`: `docker.service` 후에 이 서비스가 시작되어야 함을 의미한다.
- `[Service]`: 서비스의 동작 방식을 정의한다.
  - `RemainAfterExit=yes`: 서비스가 한 번 시작되면, 종료 상태로 전환하지 않고 활성 상태로 남아있다.
  - `ExecStart`: 서비스가 시작될 때 실행할 명령어이다. 여기서는 Docker의 `jenkins` 컨테이너를 시작한다.
  - `ExecStop`: 서비스가 종료될 때 실행할 명령어이다. 여기서는 Docker의 `jenkins` 컨테이너를 중지한다.
- `[Install]`: 이 서비스를 언제 활성화할지 정의한다.
  - `WantedBy=multi-user.target`: 다중 사용자 모드에서 이 서비스가 활성화되어야 함을 나타낸다.

```
sudo systemctl daemon-reload
```

```
sudo systemctl enable docker-jenkins.service
```

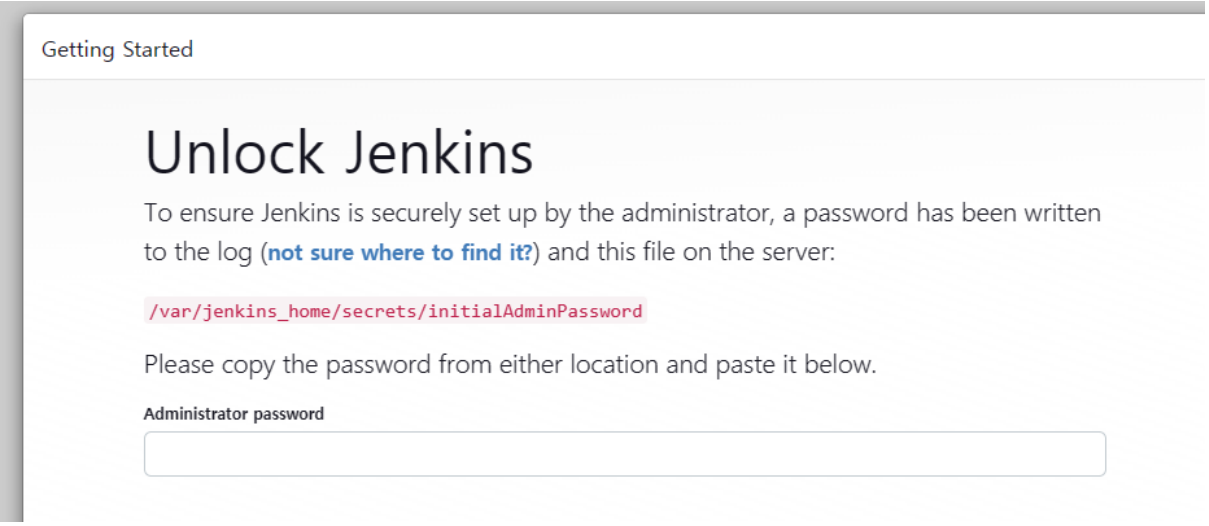
```
sudo systemctl start docker-jenkins.service
```

- 다음과 같은 명령어를 통해 시스템 부팅시 'jenkins' 컨테이너가 자동으로 시작될 것이다.

Jenkins 접속

```
http://j9c109.p.ssafy.io:8080
```

- 포워딩 및 방화벽 개방 작업까지 완료되었으니, 서버 IP 및 포트번호를 웹 브라우저에 입력하고 접속한다.
-



- 웹페이지에서는 비밀번호를 확인하기 위해 위 경로에 있다고 하지만, 이 경로는 Jenkins가 설치되어 있는 컨테이너 안의 파일이기 때문에 아래와 같은 방법을 이용한다.

```
docker exec -it jenkins /bin/bash
```

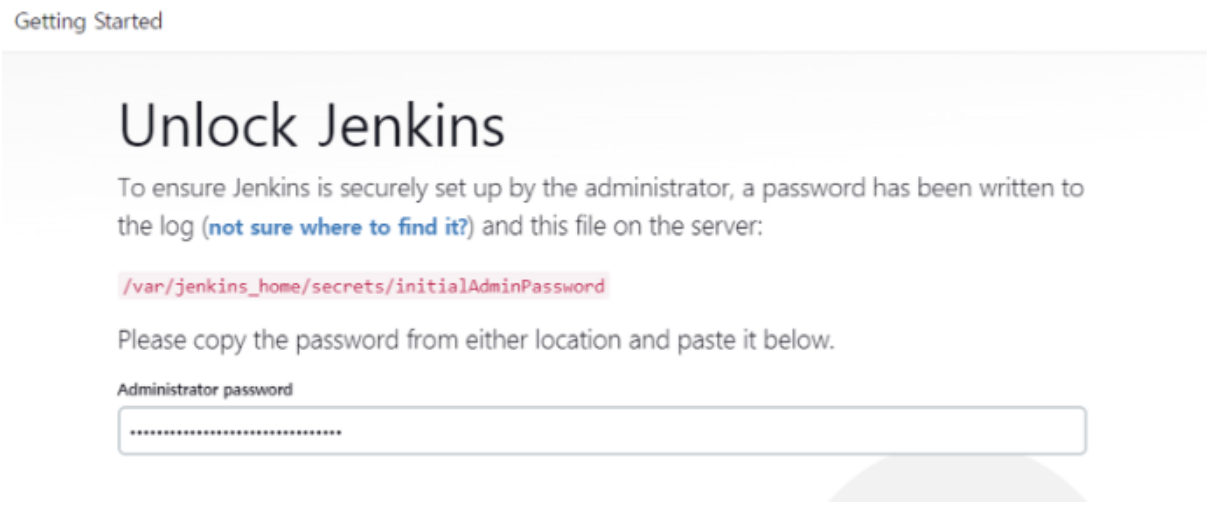
Jenkins 컨테이너 안으로 접속하기 위해 다음 명령어를 활용하여 컨테이너의 bash 쉘에 접속한다.

```
cd /var/jenkins_home/secrets
```

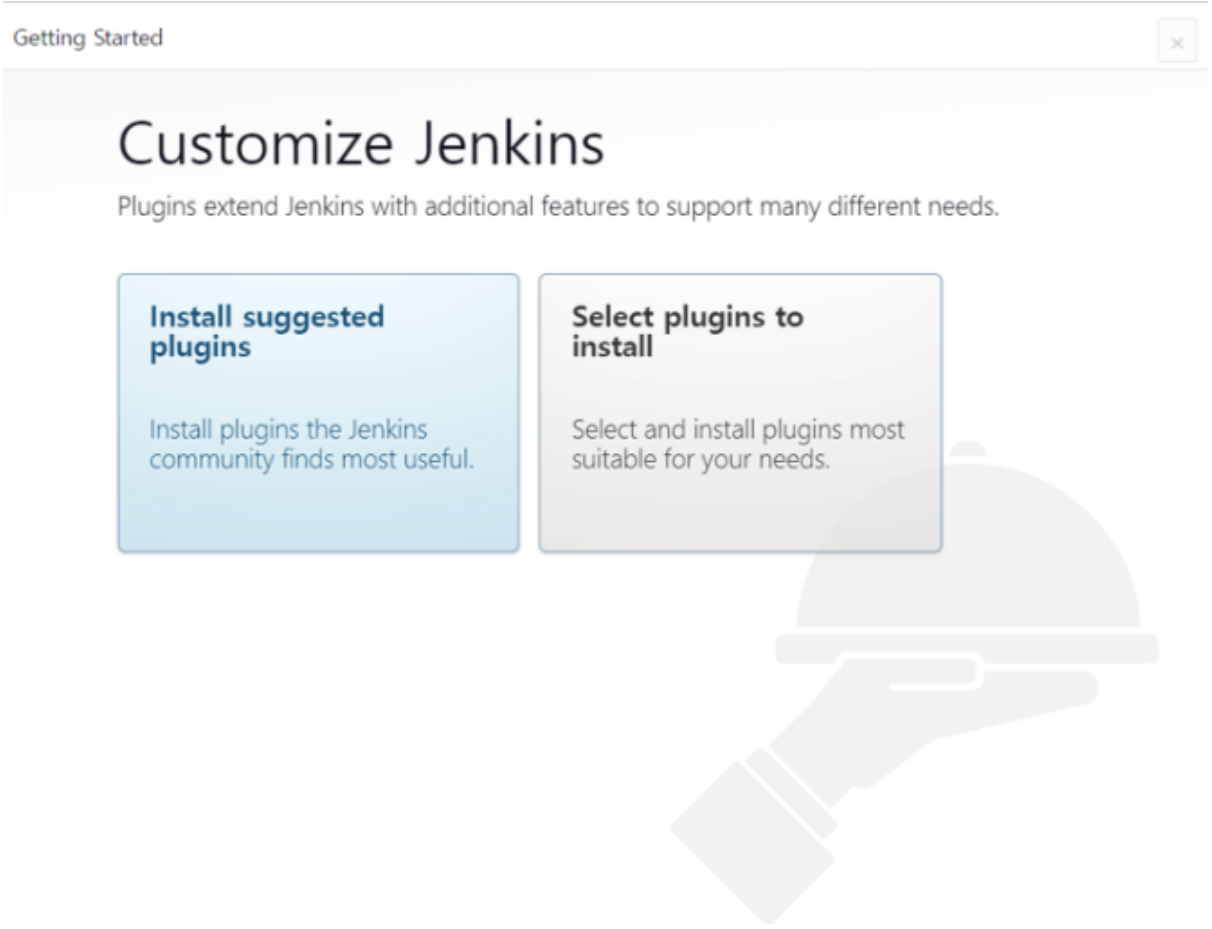
```
cat initialAdmiinPassword
```

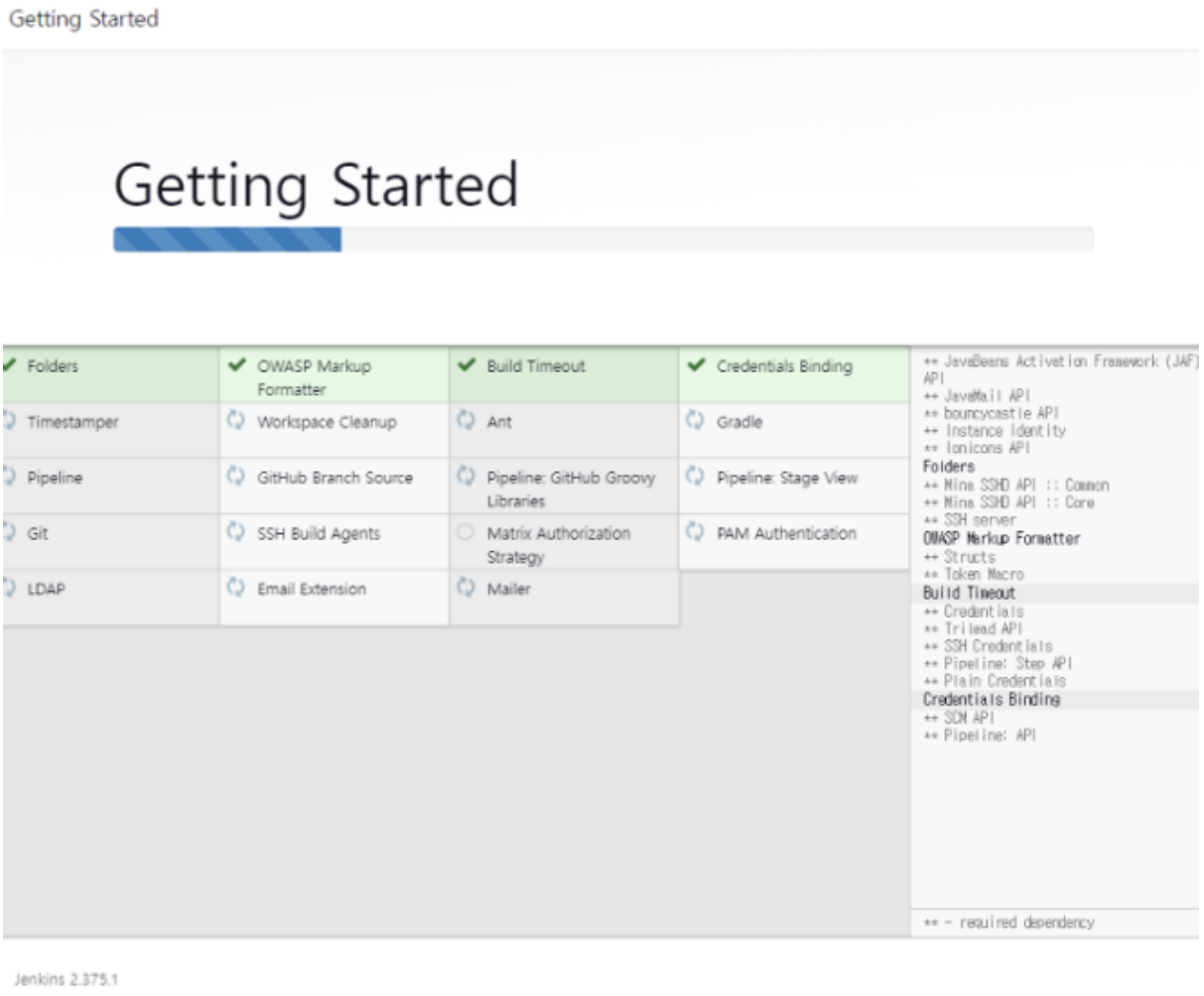
```
root@267887cacf12:/# cd /var/jenkins_home/secrets
root@267887cacf12:/var/jenkins_home/secrets# ls -al
total 20
drwx------  2 root root 4096 Jan  8 08:36 .
drwxr-xr-x 11 root root 4096 Jan  8 08:37 ..
-rw-r-----  1 root root   33 Jan  8 08:36 initialAdminPassword
-rw-r--r--  1 root root   32 Jan  8 08:36 jenkins.model.Jenkins.crumbSalt
-rw-r--r--  1 root root  256 Jan  8 08:36 master.key
root@267887cacf12:/var/jenkins_home/secrets# vi initialAdminPassword
```

- cat 명령어를 이용하여 초기 비밀번호를 확인한다. 이후 exit 를 하여 컨테이너의 bash 쉘에서 나가 Host OS의 쉘로 복귀한다.



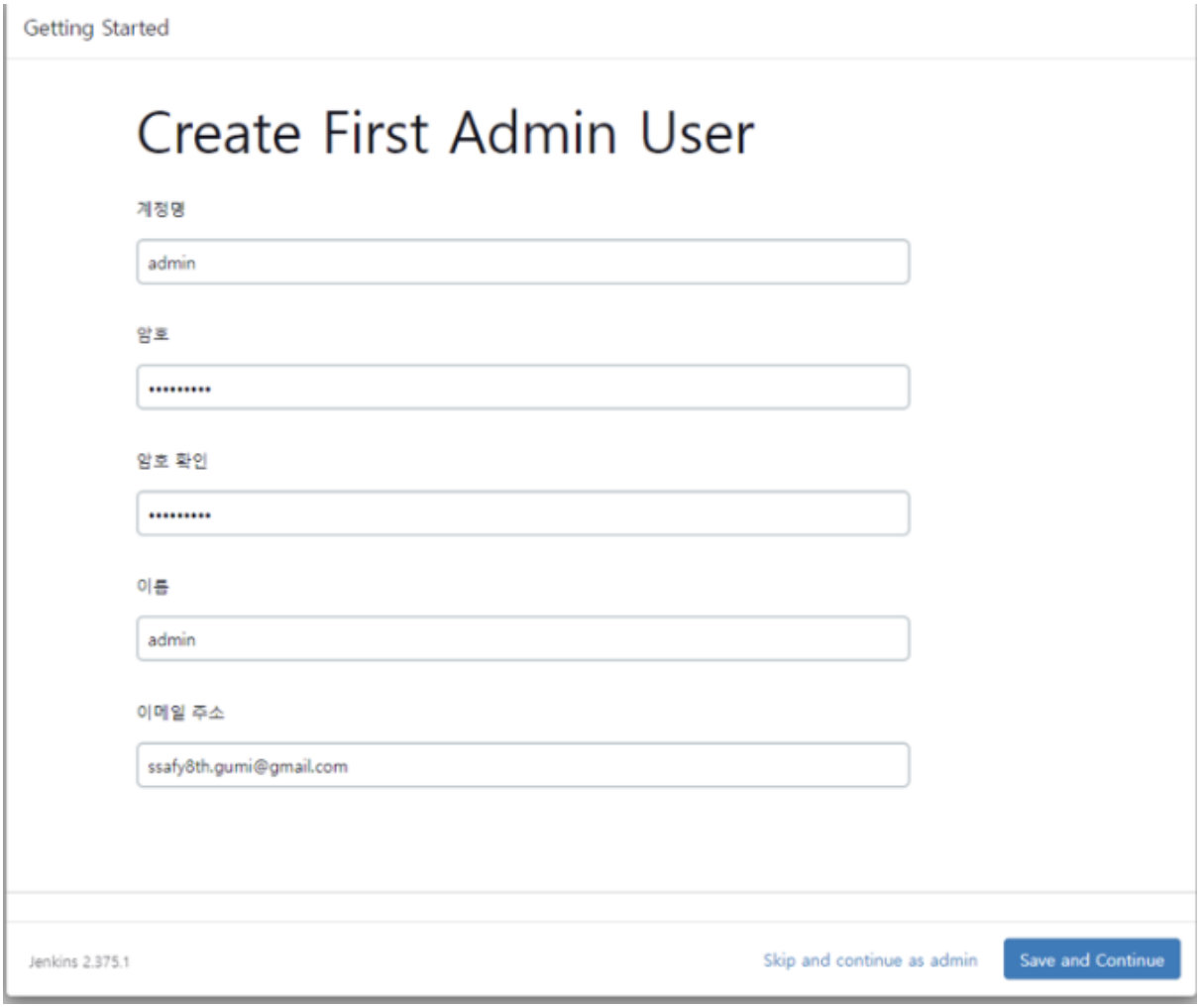
- 조금 전 확인한 방법으로 얻어낸 관리자 비밀번호를 입력한다

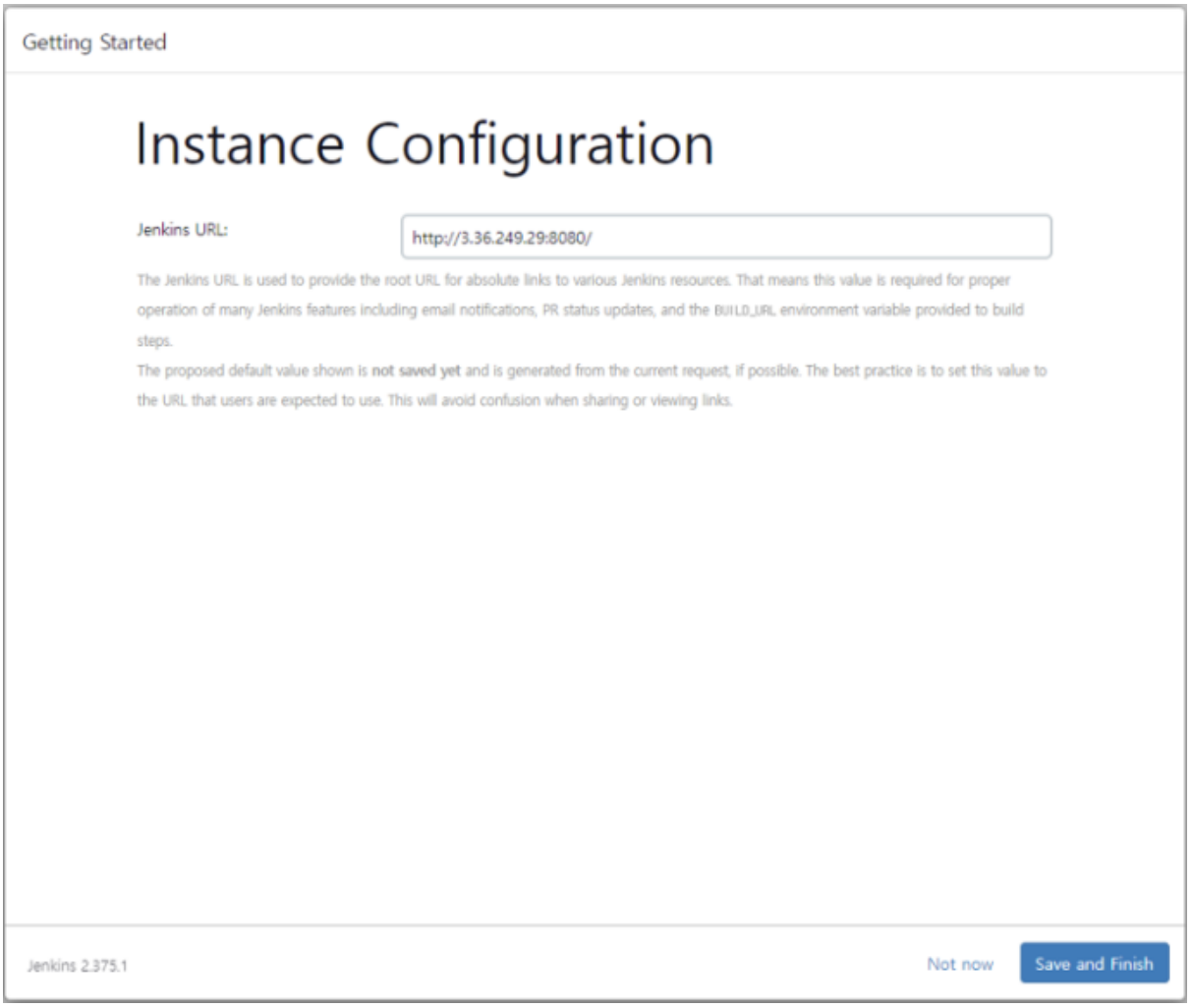

Getting Started
Customize Jenkins
Plugins extend Jenkins with additional features to support many different needs.
Install suggested plugins
Install plugins the Jenkins community finds most useful.
Select plugins to install
Select and install plugins most suitable for your needs.

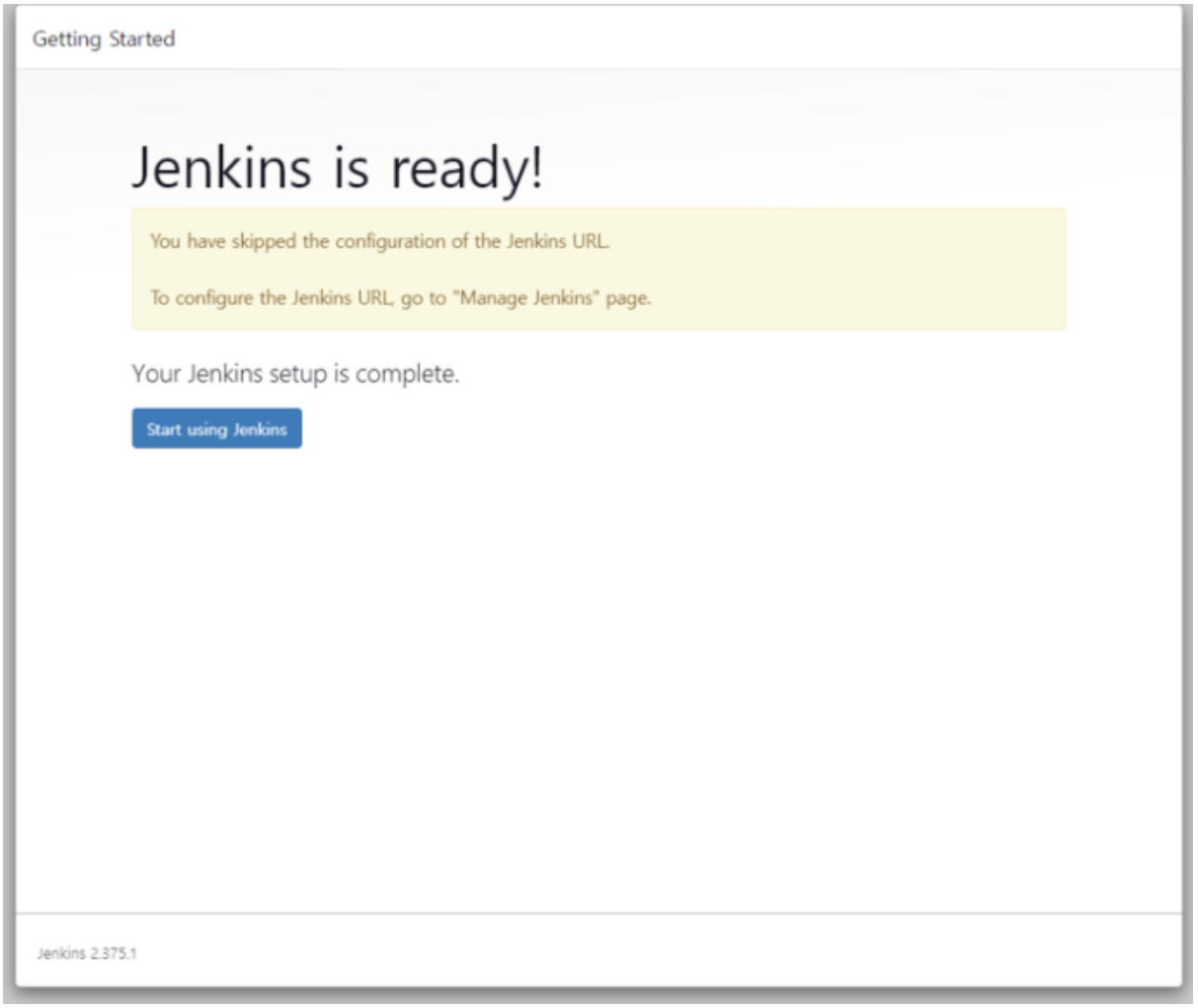


기본 플러그인 설치

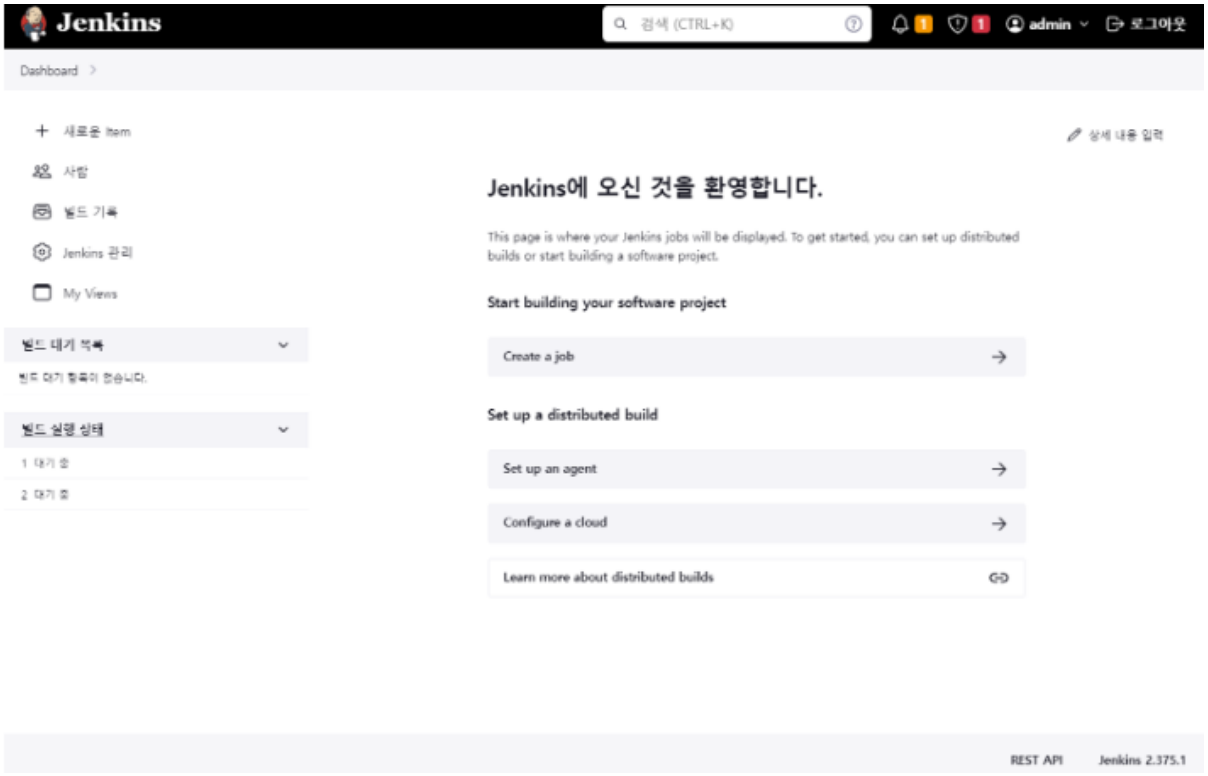
- 초기 페이지에 접속하면, 우선 추천하는 플러그인을 설치한다 **(Install suggested plugins)**


Getting Started
Create First Admin User
계정명
admin
암호
••••••••••
암호 확인
••••••••••
이름
admin
이메일 주소
ssafy8th.gumi@gmail.com
Jenkins 2.375.1
Skip and continue as admin
Save and Continue


Getting Started
Instance Configuration
Jenkins URL:
http://3.36.249.29:8080/
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.
Jenkins 2.375.1
Not now
Save and Finish



- 기타 jenkins 설정을 마무리한다.



Jenkins 내부에 Docker 패키지 설치(컨테이너 시작마다 하는 작업)

```
docker exec -it jenkins /bin/bash
```

- jenkins 컨테이너 접속

```
apt-get update && apt-get -y install apt-transport-https ca-certificates curl gnupg2 s
oftware-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 Repository 등록 및 docker-ce 패키지 설치

▼ 명령어 상세 설명

- `apt-get update`: 패키지 인덱스를 업데이트한다.

- `apt-get -y install apt-transport-https ca-certificates curl gnupg2 software-properties-common`: HTTPS를 통한 패키지 다운로드와 추가적인 소프트웨어 속성을 관리할 수 있는 필수 패키지들을 설치한다.
- `curl -fsSL https://download.docker.com/linux/$(. /etc/os-release; echo "$ID")/gpg > /tmp/dkey; apt-key add /tmp/dkey`: Docker의 GPG 키를 다운로드해서 APT 키링에 추가한다.
- `add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/$(. /etc/os-release; echo "$ID") $(lsb_release -cs) stable"`: Docker의 APT 리포지토리를 시스템에 추가한다.
- `apt-get update`: 리포지토리를 추가한 후 다시 패키지 인덱스를 업데이트한다.
- `apt-get -y install docker-ce`: Docker CE를 설치한다.

```
groupadd -f docker
usermod -aG docker jenkins
chown root::docker /var/run/docker.sock
```

- 이 설정을 통해 jenkins User는 Docker 커맨드를 실행할 때 Sudo를 사용할 필요가 없게 된다. 이는 Jenkins와 같은 CI/CD 툴에서 Docker를 더 쉽게 사용할 수 있게 해준다.

▼ 명령어 상세 설명

1. `groupadd -f docker`: `docker`라는 이름의 그룹을 만든다. `f` 옵션은 그룹이 이미 존재할 경우에는 아무 작업도 수행하지 않는다.
2. `usermod -aG docker jenkins`: `jenkins` 사용자를 `docker` 그룹에 추가한다. `aG` 옵션은 사용자를 그룹에 추가할 때 사용되며, 이 사용자는 이제 `docker` 명령을 비-루트 사용자로 실행할 수 있다.
3. `chown root::docker /var/run/docker.sock`: Docker 소켓 파일(`/var/run/docker.sock`)의 소유자를 `root`로, 그룹을 `docker`로 변경한다. 이렇게 하면 `docker` 그룹의 멤버는 Docker 데몬에 접근할 수 있다.

Jenkins 파이프라인 설정

플러그인 설치 목록

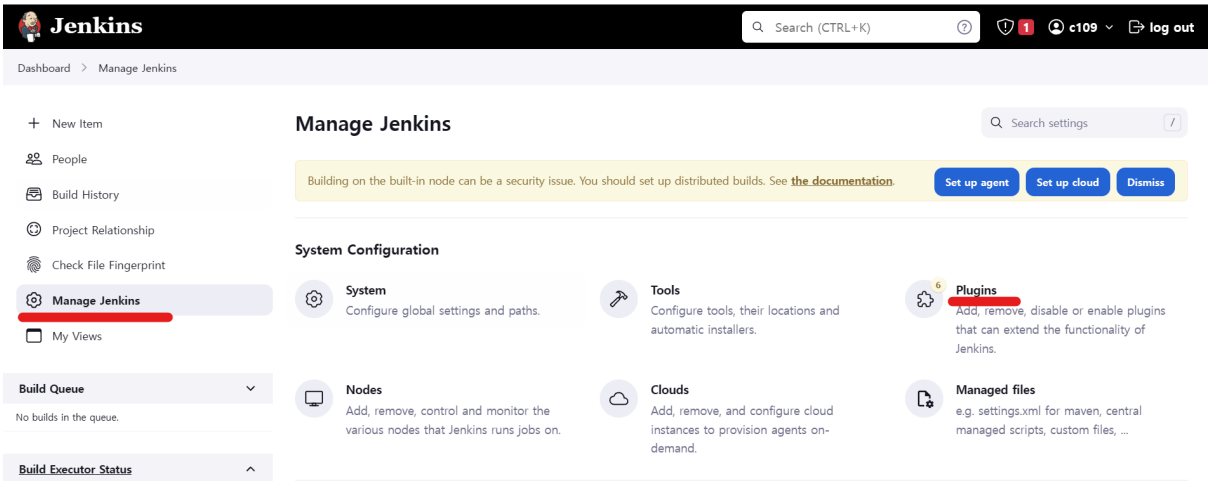
```
# ssh 커맨드 입력에 사용
SSH Agent

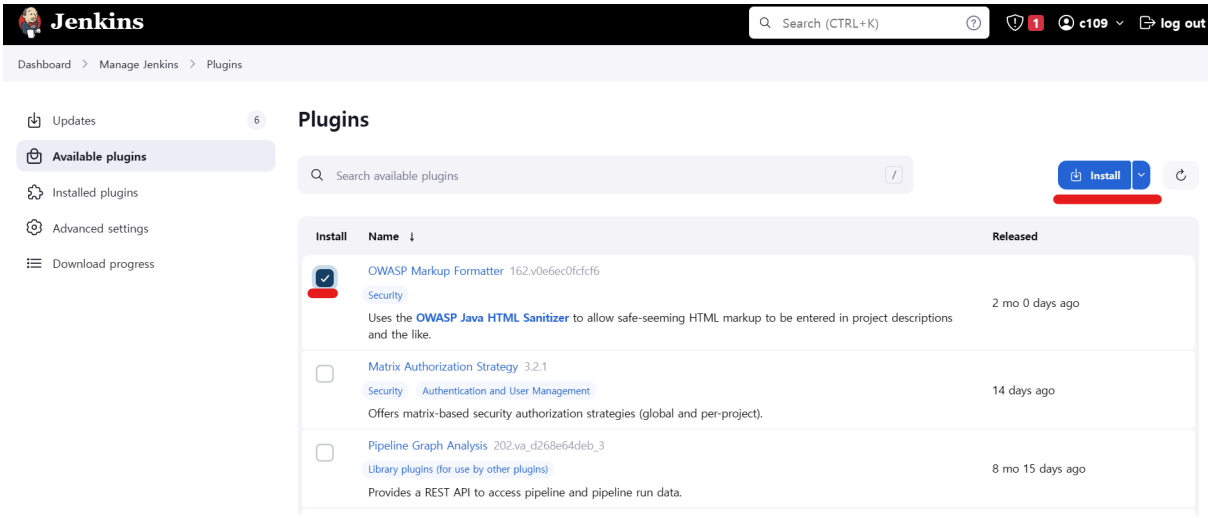
# docker 이미지 생성에 사용
Docker
Docker Commons
Docker Pipeline
Docker API

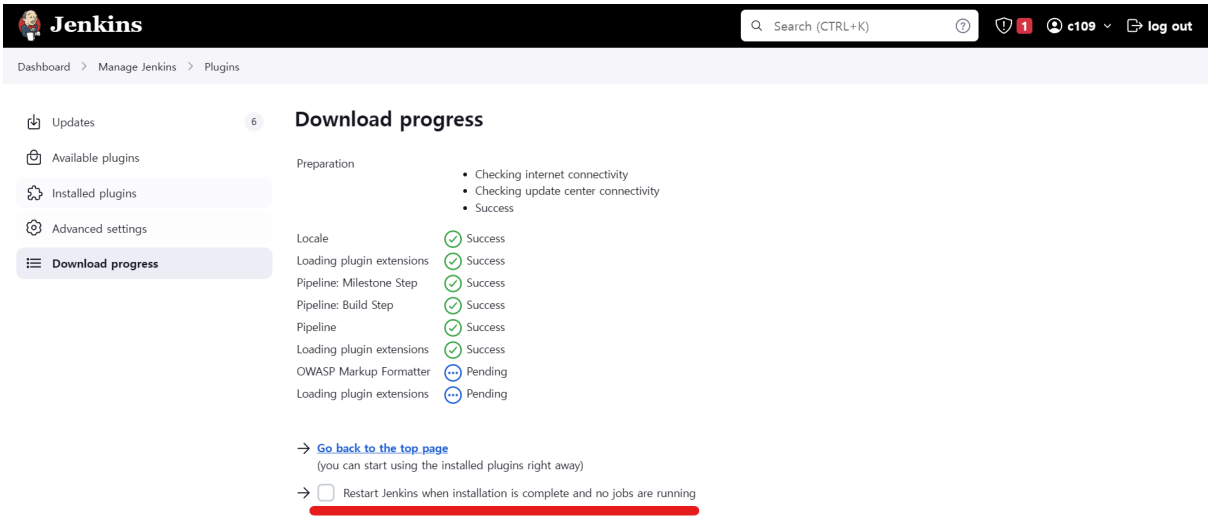

# 타사 레포지토리 이용시 사용 (GitLab, Github 등)
GitLab
GitLab API
GitLab Authentication

# Node.js 빌드시 사용
NodeJS

#웹훅 트리거
Generic Webhook Trigger
```

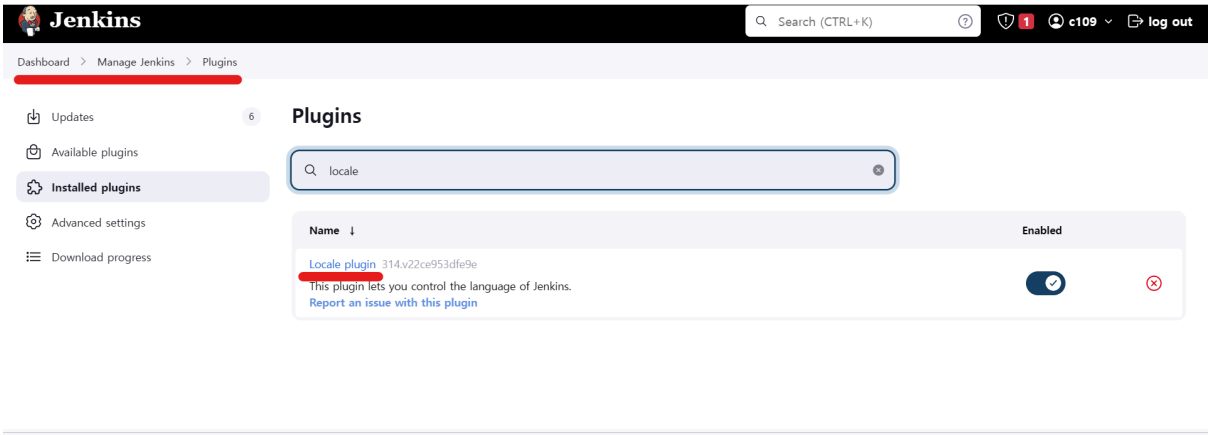


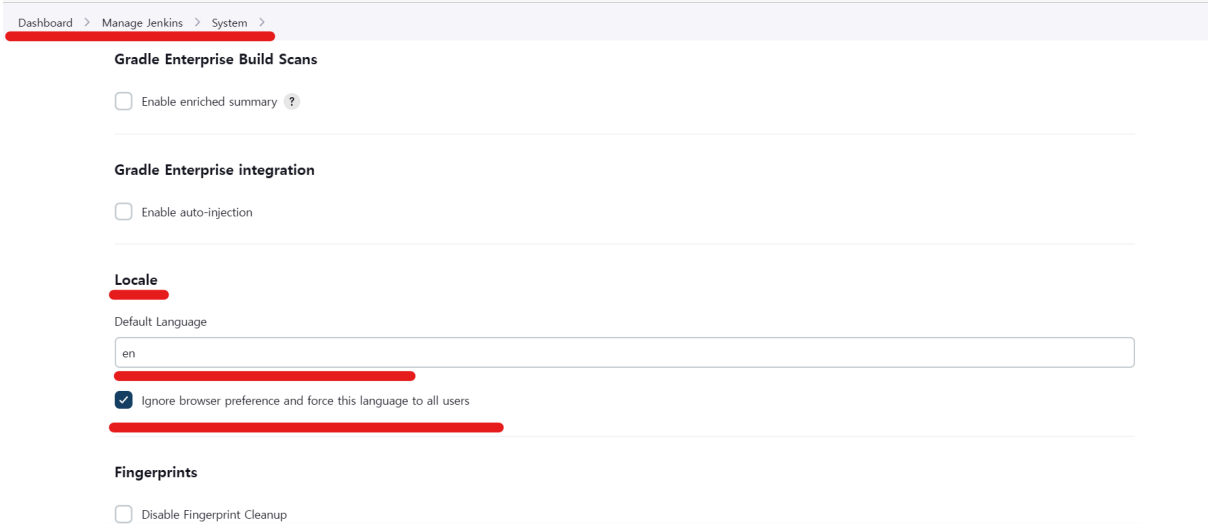




- Avaliable plugins 선택 후 설치하고자 하는 플러그인 이름 검색
- 설치할 플러그인 선택 후 **Install without restart** 클릭
- 알아서 설치 완료. 이후 **메인 페이지로 돌아가기** 클릭

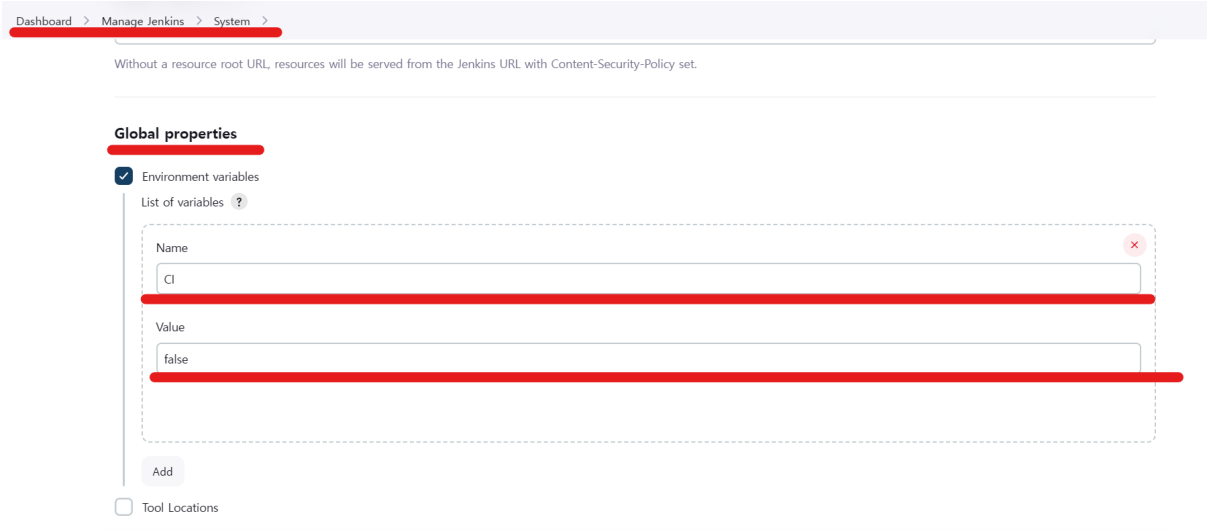
Jenkins 로케일 언어(locale language) 설정





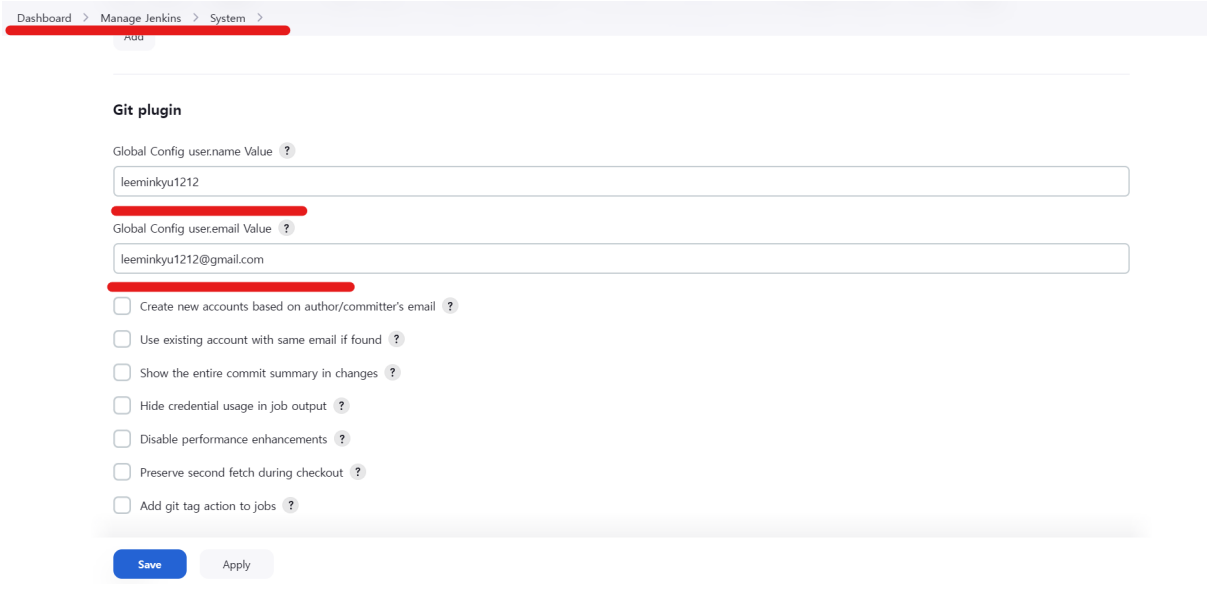
- Jenkins에서 국문/영문 혼용이 너무 심해 차라리 영문으로 사용하는게 편리하다.
- Jenkins 관리 → Plugin Manger → Avaliable plugins → Locale 검색 → Install without restart
- Dashboard → Jenkins 관리 → System → Locale
  Default Language: en , Ignore browser preference and force language to all users 체크박스 체크
- 저장

Jenkins 환경변수 설정(Node.js 빌드시 사용된다.)



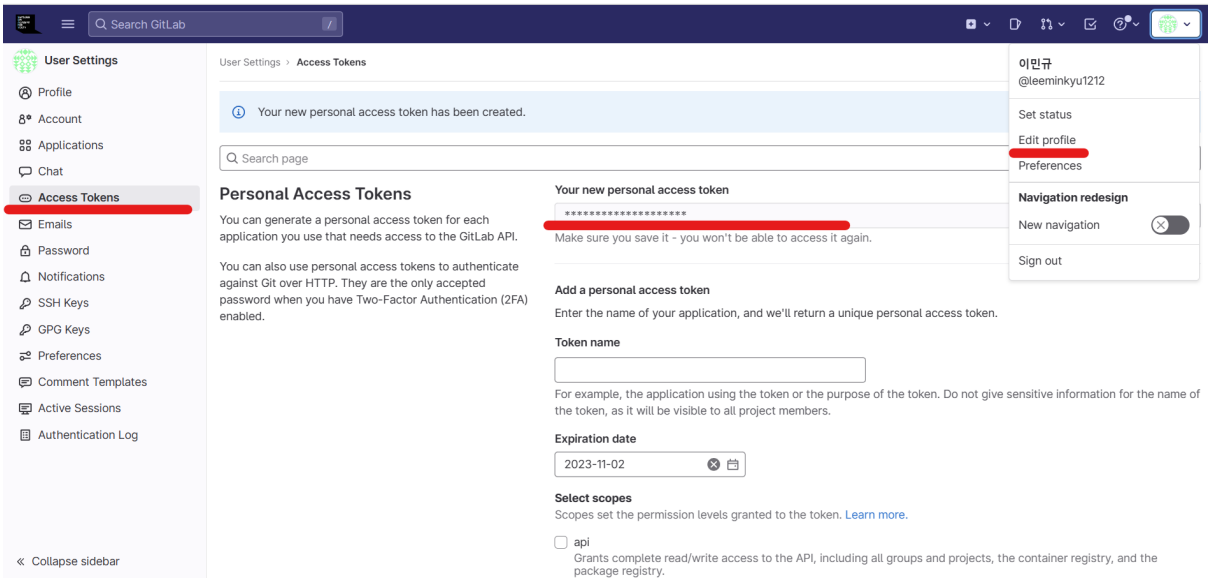
- Manage Jenkins → System →Global Properties
  - Environetn variables 체크
  - CI, false 환경변수 추가
  - Save 클릭

Jenkins 에 user name, email 기입



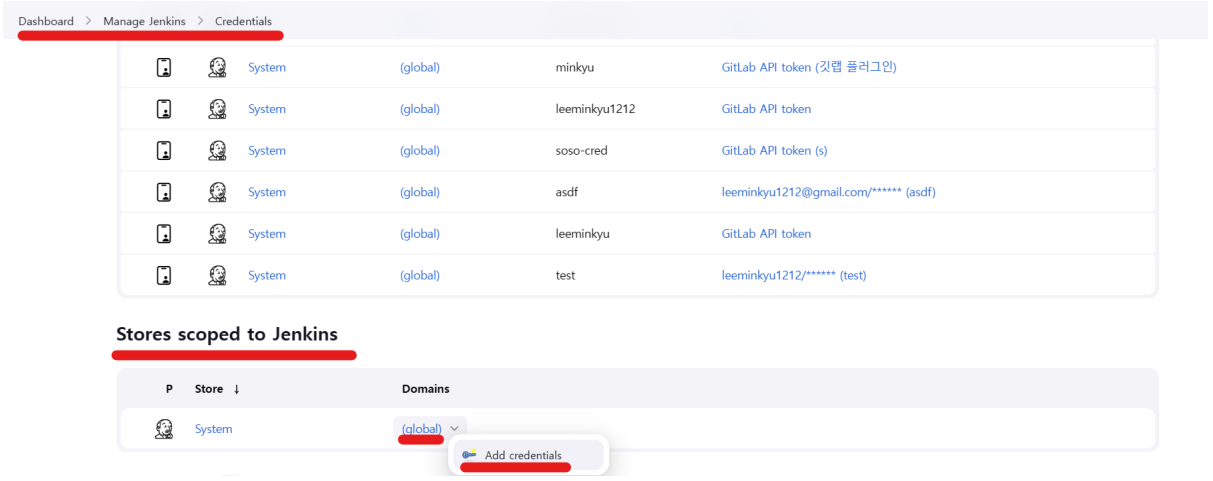
- DashBoard→ Manage Jenkins → System→ Git plugin
- git username ,git email을 기입해준다.

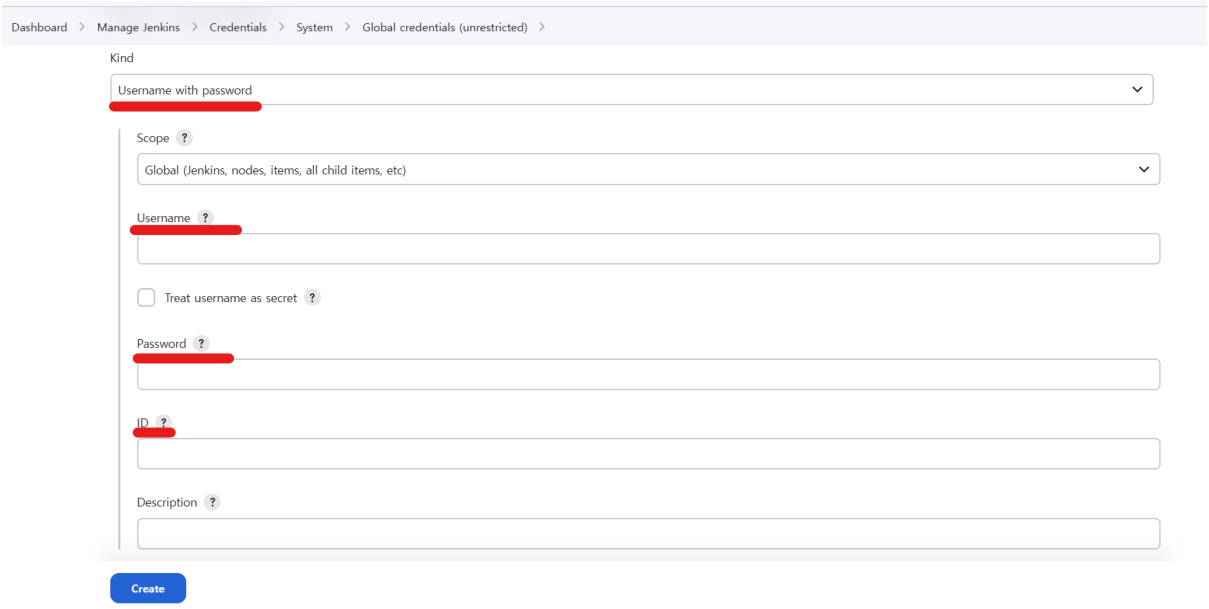
Gitlab AccessToken 생성



lab.ssafy.com에 로그인후 Edit Profile → User Settings → Access token 에서 토큰이름, 만료일,허용 권한 등을 체크한 뒤에 발급을 받는다.

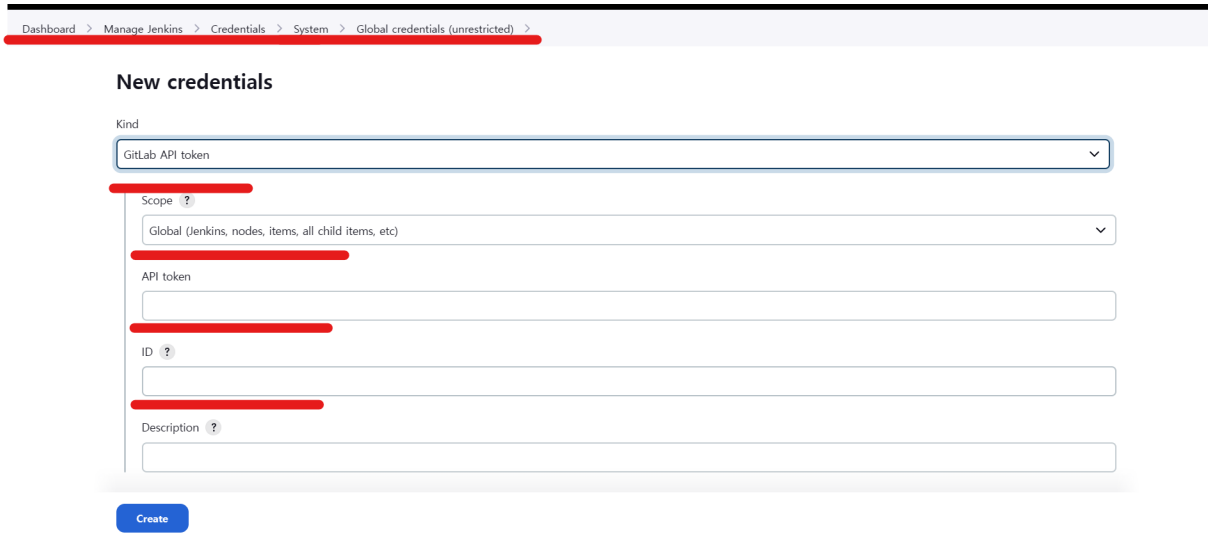
GitLab Credential 등록 (Username with password)





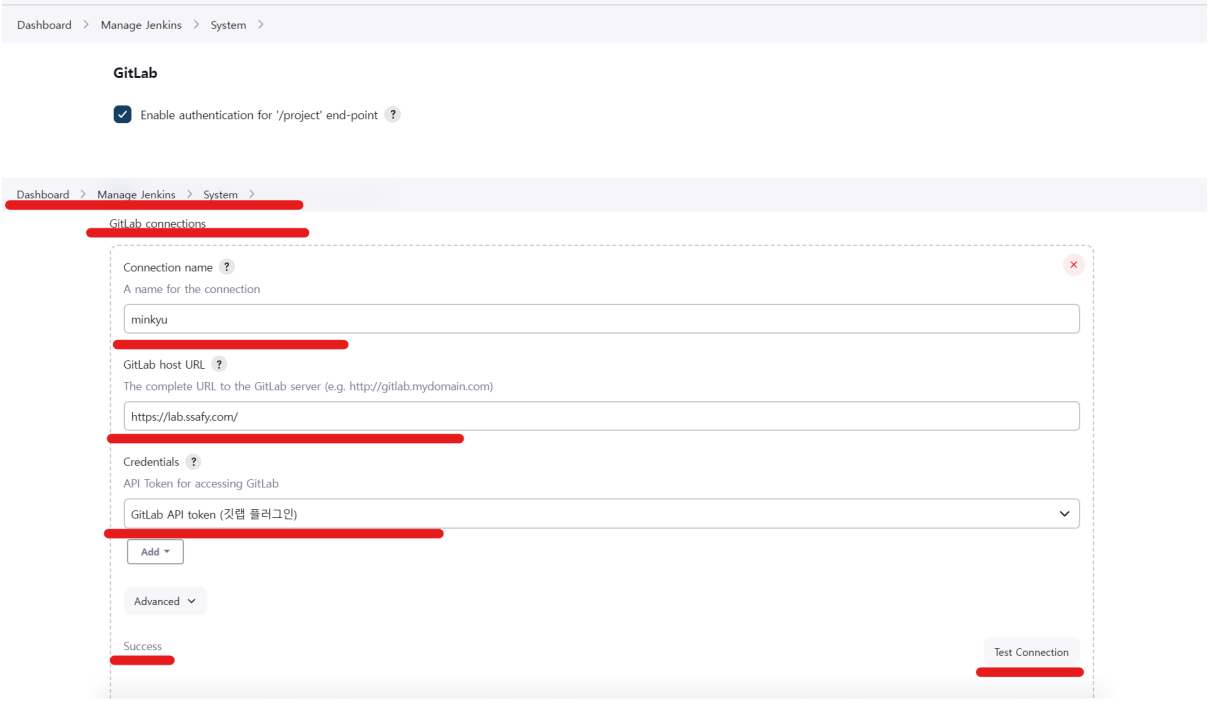
- DashBoard→ Manage Jenkins → Credentials → Stores scoped to Jenkins → domains → (global)→ add Credentials
- 정보 입력 후 **Create** 클릭
  - Kind : Username with password 선택
  - Username : Gitlab 계정 아이디 입력
  - Password : Gitlab 계정 비밀번호 입력 **(토큰 발행시, API 토큰 입력)**
  - ID : Credential에 대한 별칭

GitLab Credential 등록 (API Token)



- Jenkins 관리 - Manage Jenkins → Credentials → System → Global Credentials→ New Credentials
- Kind : Gitlab API token 선택
- API tokens : Gitlab 계정 토큰 입력
- ID : Credential에 대한 별칭

GitLab 커넥션 추가

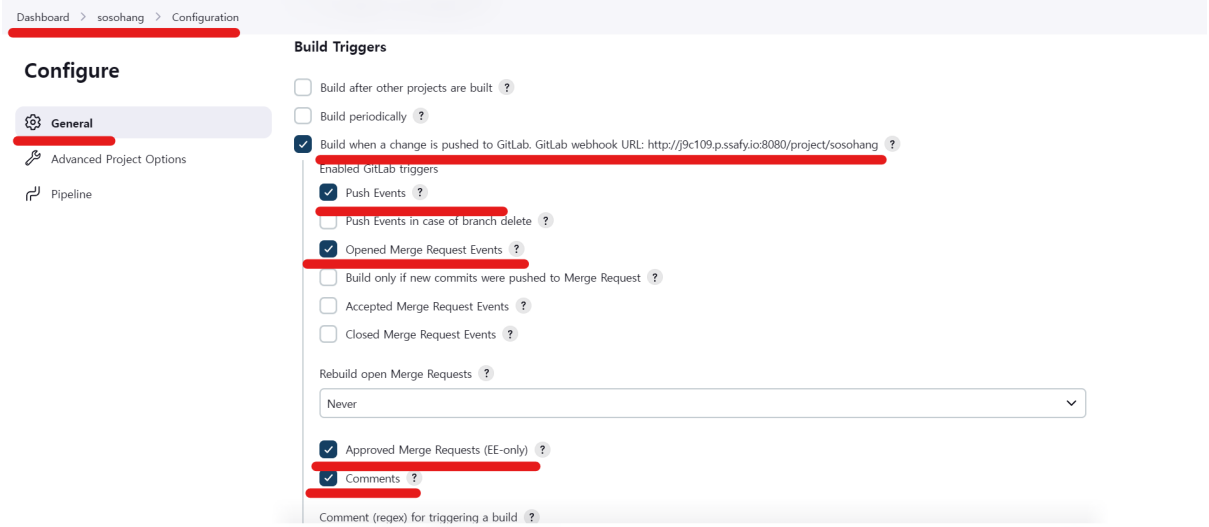


- Manage Jenkins → System → GitLab connection

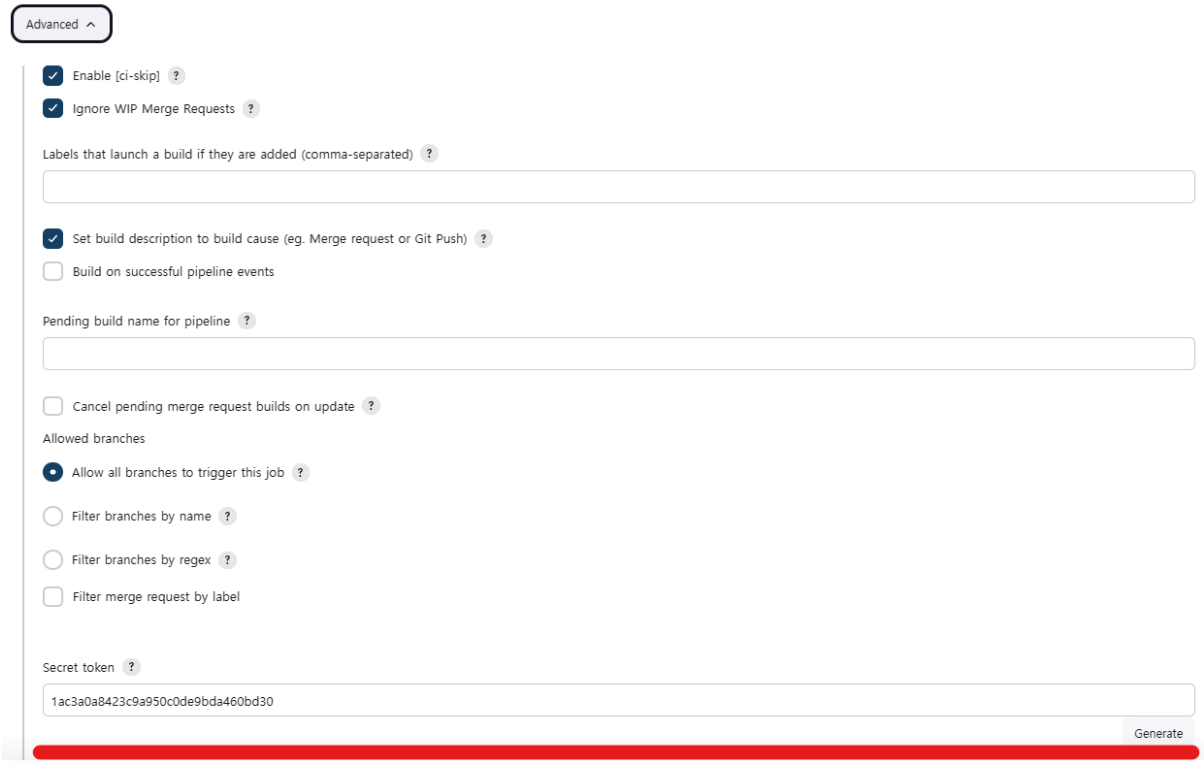
- Gitlab의 **Enable authentication for '/project' end-point** 체크
  - Connection name : Gitlab 커넥션 이름 지정
  - Gitlab host URL : Gitlab 시스템의 Host 주소 입력
  - Credentials : 조금 전 등록한 **Jenkins Credential (API Token)**을 선택
  - 이후, **Test Connection**을 눌러 Success가 뜨면 **저장** 클릭
    - 아니라면 입력한 정보를 다시 확인

Jenkins Webhook Integration 설정

- GitLab에서 event가 발생하면, CI/CD 를 진행하기 위해 Webhook을 설정을 하는 과정



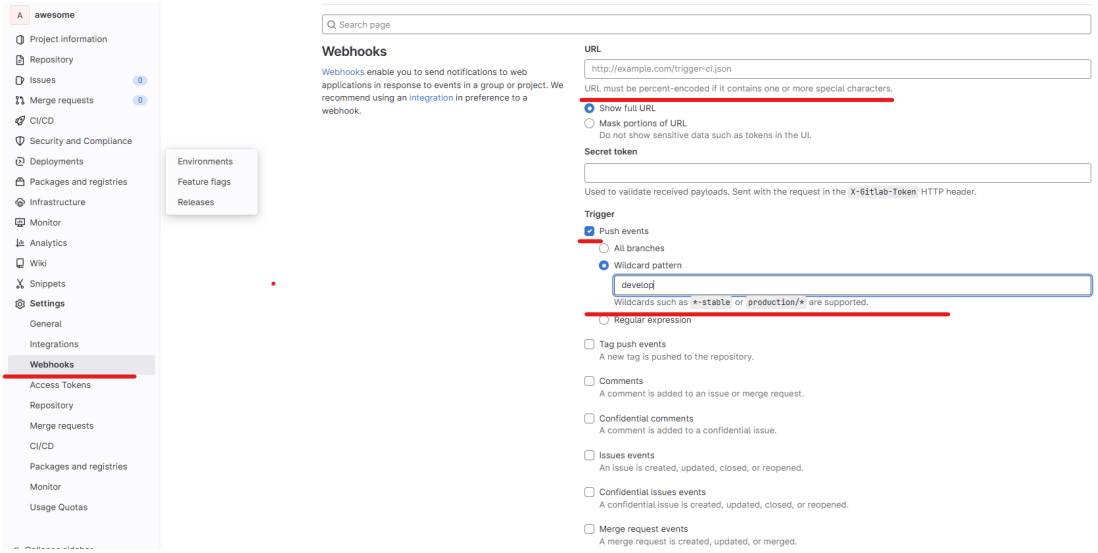
- Pipeline 아이템에 다음과 같은 설정 추가
  - General - Build Triggers
    - Build when a change is pushed to Gitlab 체크
    - Push Events 체크
    - Opened Merge Request Events 체크
    - Approved Merge Request (EE-only) 체크
    - Comments 체크



Advanced→ Generate 클릭

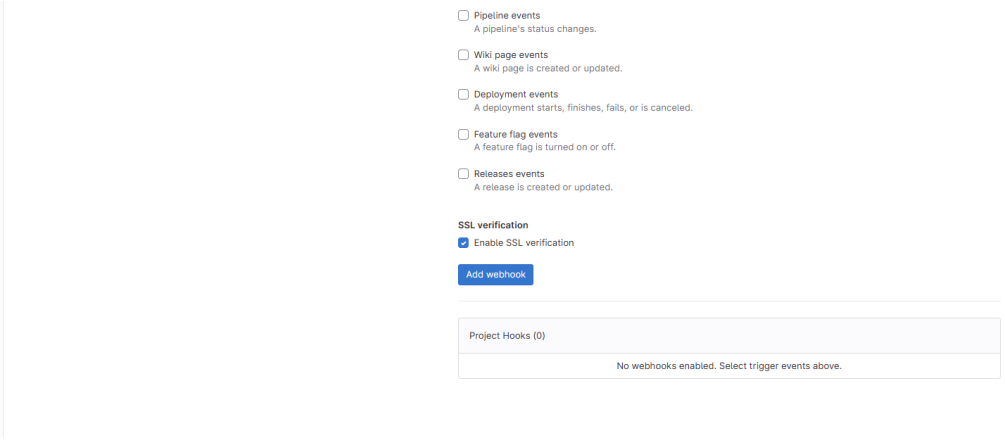
- 발행된 Secret token 복사후 Save

Gitlab Webhook



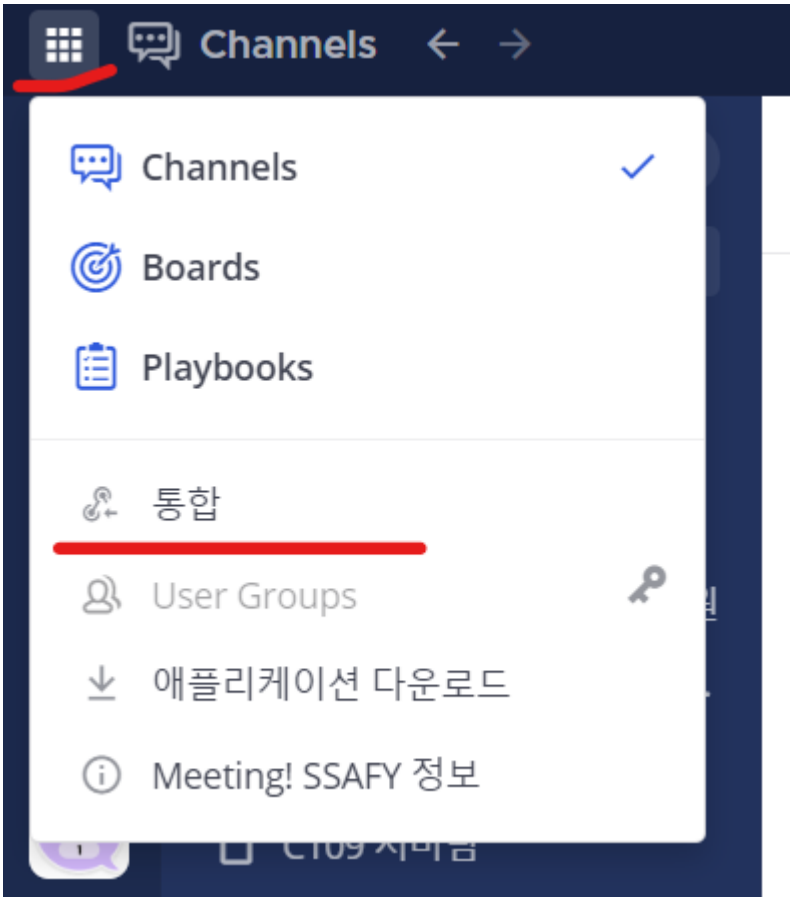
- Gitlab 프로젝트 → Settings → Webhooks
  - jenkins 접속 URL을 기입

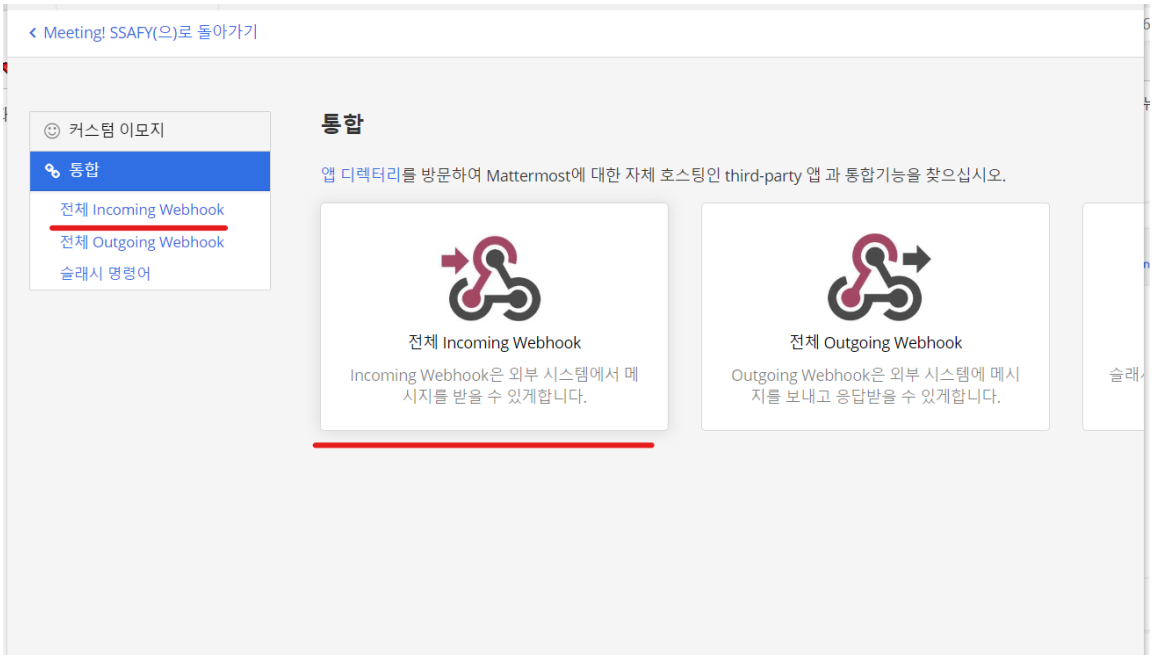
- Secret Token에 복사한 토큰 기입
- Trigger → Push events
- develop 브랜치에 push 이벤트가 발생하면 빌드를 진행하기 위해 Wildcard patterns에 develop 추가
-

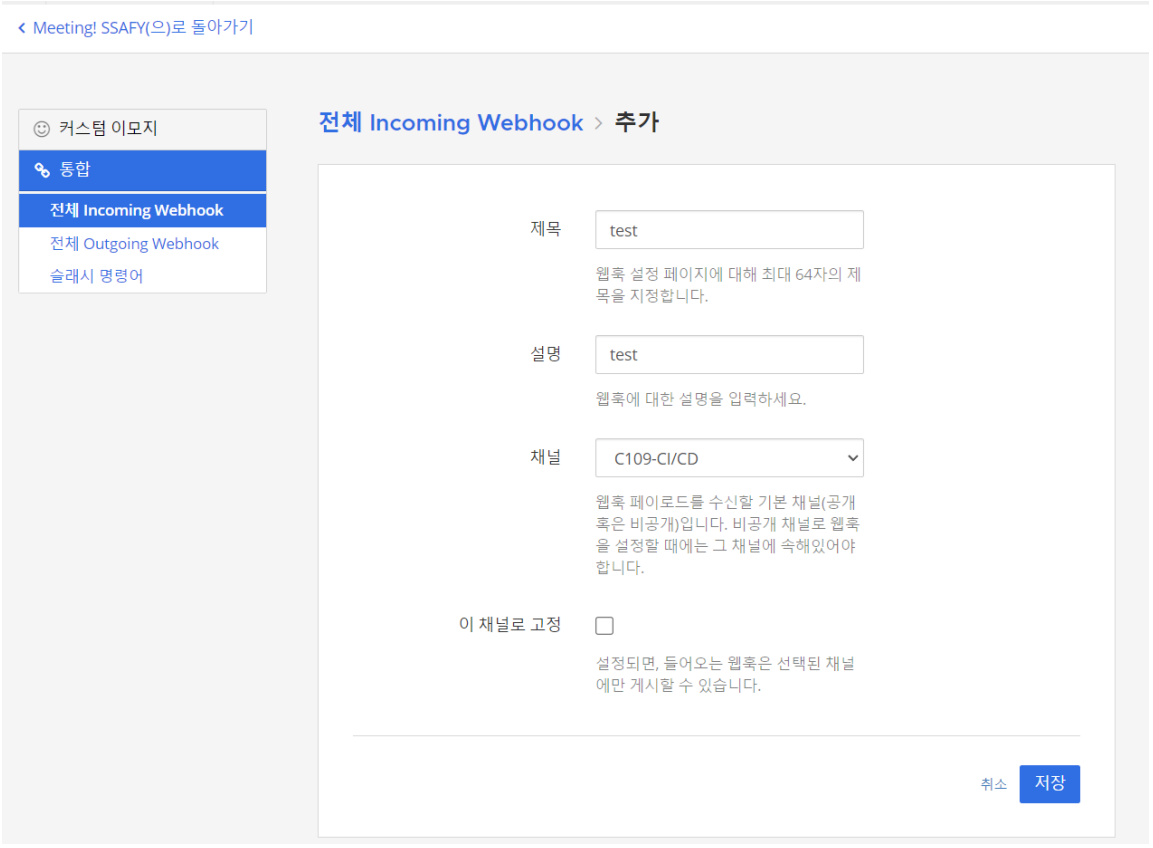


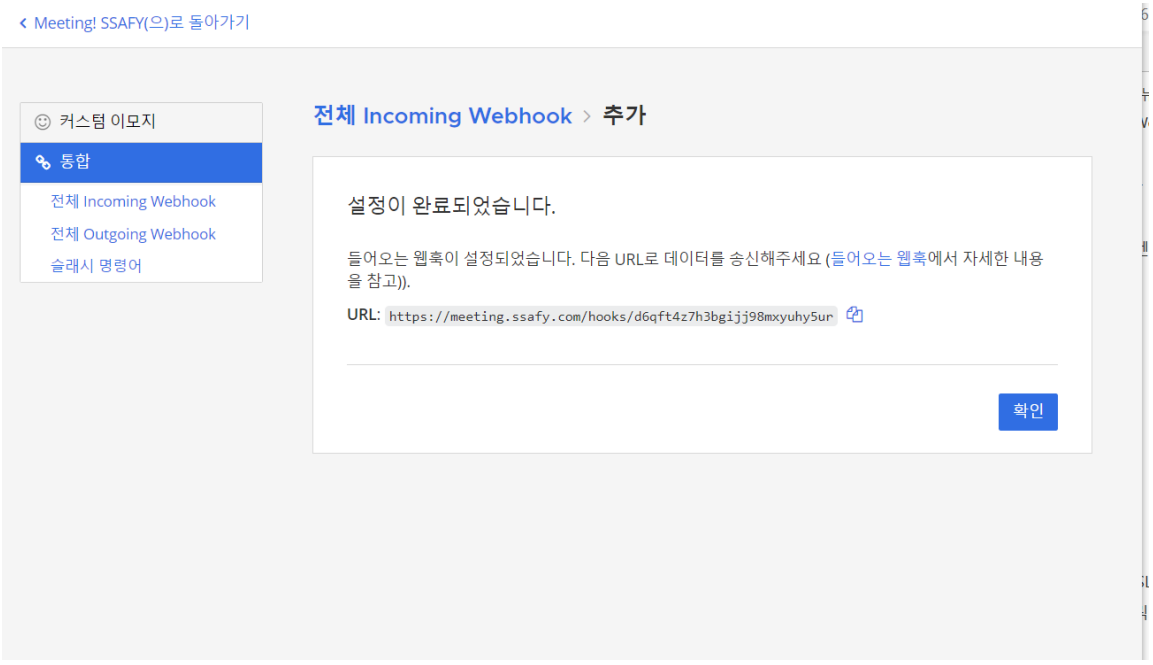
- SSL verification의 Enable SSL verifiacation 체크
  - 이후 Add webhook 클릭

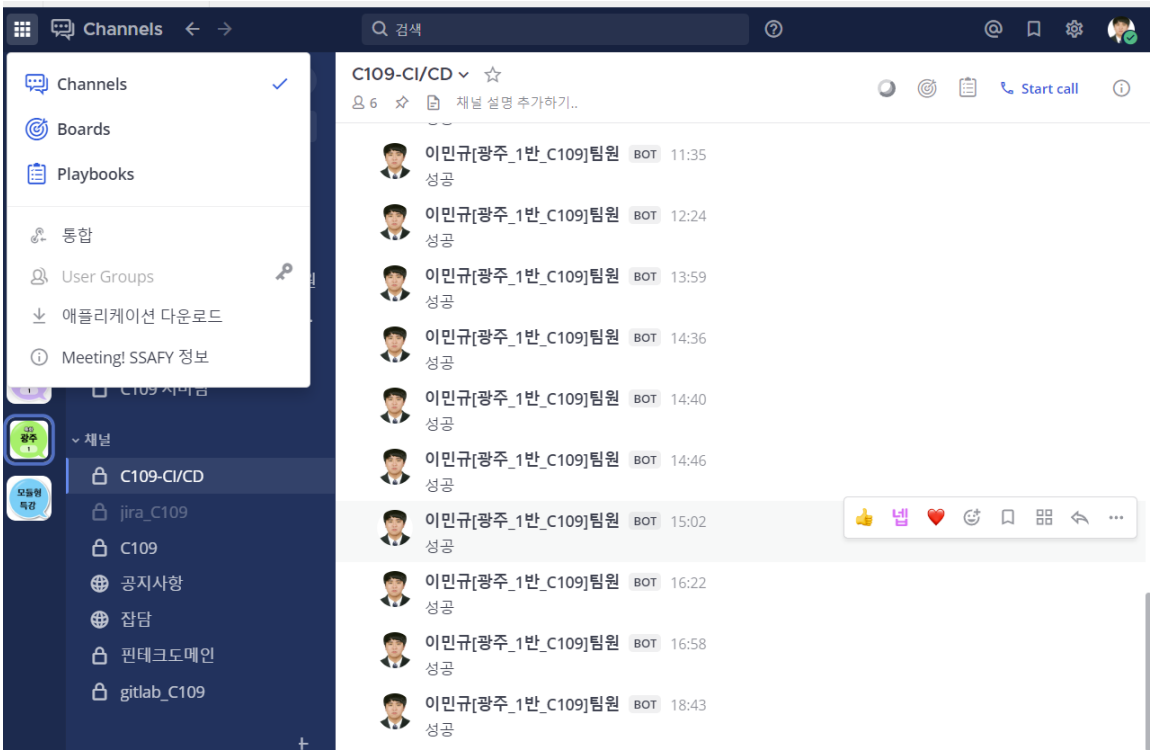
Mattermost에 배포 완료후 메시지 보내기


Channels
Channels
Boards
Playbooks
통합
User Groups
애플리케이션 다운로드
Meeting! SSAFY 정보


Meeting! SSAFY(으)로 돌아가기
커스텀 이모지
통합
전체 Incoming Webhook
전체 Outgoing Webhook
슬래시 명령어
통합
앱 디렉터리를 방문하여 Mattermost에 대한 자체 호스팅인 third-party 앱 과 통합기능을 찾으십시오.
전체 Incoming Webhook
Incoming Webhook은 외부 시스템에서 메시지를 받을 수 있게합니다.
전체 Outgoing Webhook
Outgoing Webhook은 외부 시스템에 메시지를 보내고 응답받을 수 있게합니다.


Meeting! SSAFY(으)로 돌아가기
커스텀 이모지
통합
전체 Incoming Webhook
전체 Outgoing Webhook
슬래시 명령어
전체 Incoming Webhook › 추가
제목
test
웹훅 설정 페이지에 대해 최대 64자의 제목을 지정합니다.
설명
test
웹훅에 대한 설명을 입력하세요.
채널
C109-CI/CD
웹훅 페이로드를 수신할 기본 채널(공개 혹은 비공개)입니다. 비공개 채널로 웹훅을 설정할 때에는 그 채널에 속해있어야 합니다.
이 채널로 고정
설정되면, 들어오는 웹훅은 선택된 채널에만 게시할 수 있습니다.
취소
저장

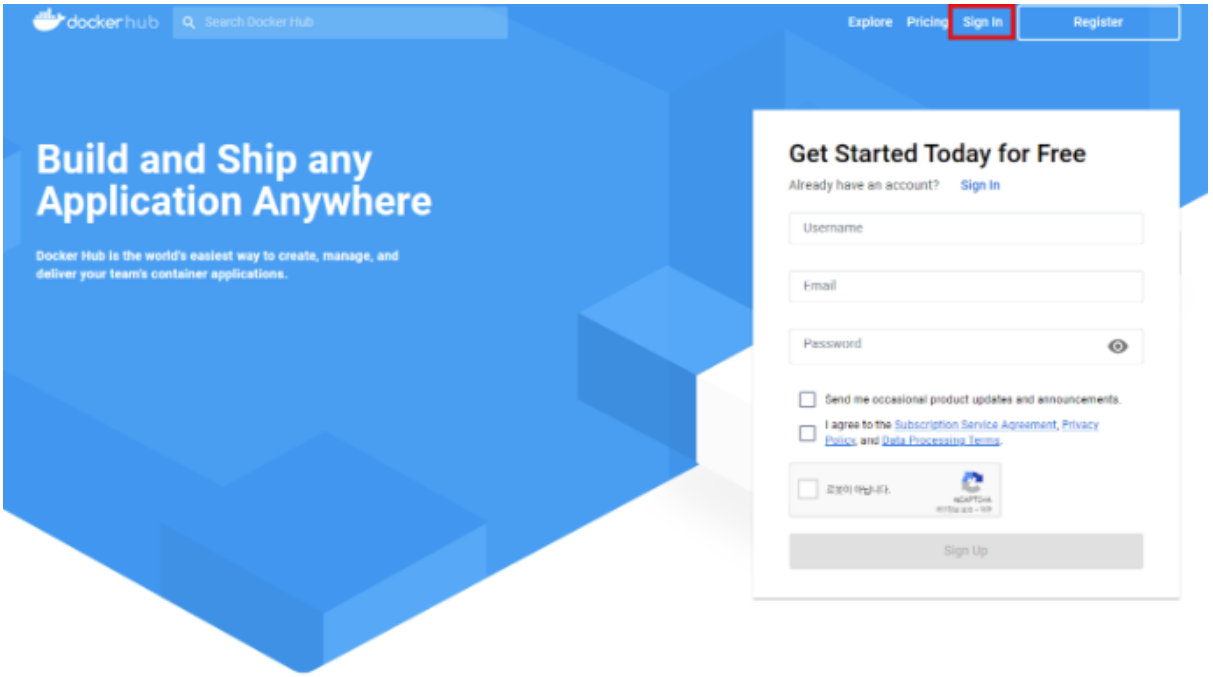

Meeting! SSAFY(으)로 돌아가기
커스텀 이모지
통합
전체 Incoming Webhook
전체 Outgoing Webhook
슬래시 명령어
전체 Incoming Webhook › 추가
설정이 완료되었습니다.
들어오는 웹훅이 설정되었습니다. 다음 URL로 데이터를 송신해주세요 (들어오는 웹훅에서 자세한 내용을 참고)).
URL: https://meeting.ssafy.com/hooks/d6qft4z7h3bgijj98mxyuhy5ur
확인



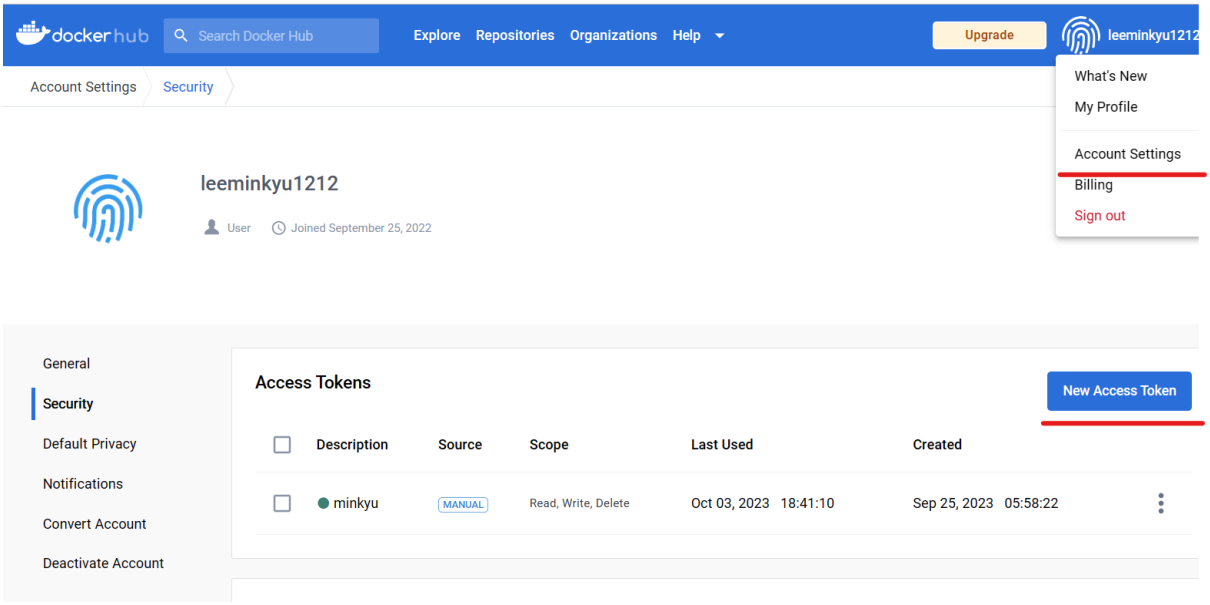
Docker Hub 토큰 생성

Docker Hub Container Image Library | App Containerization

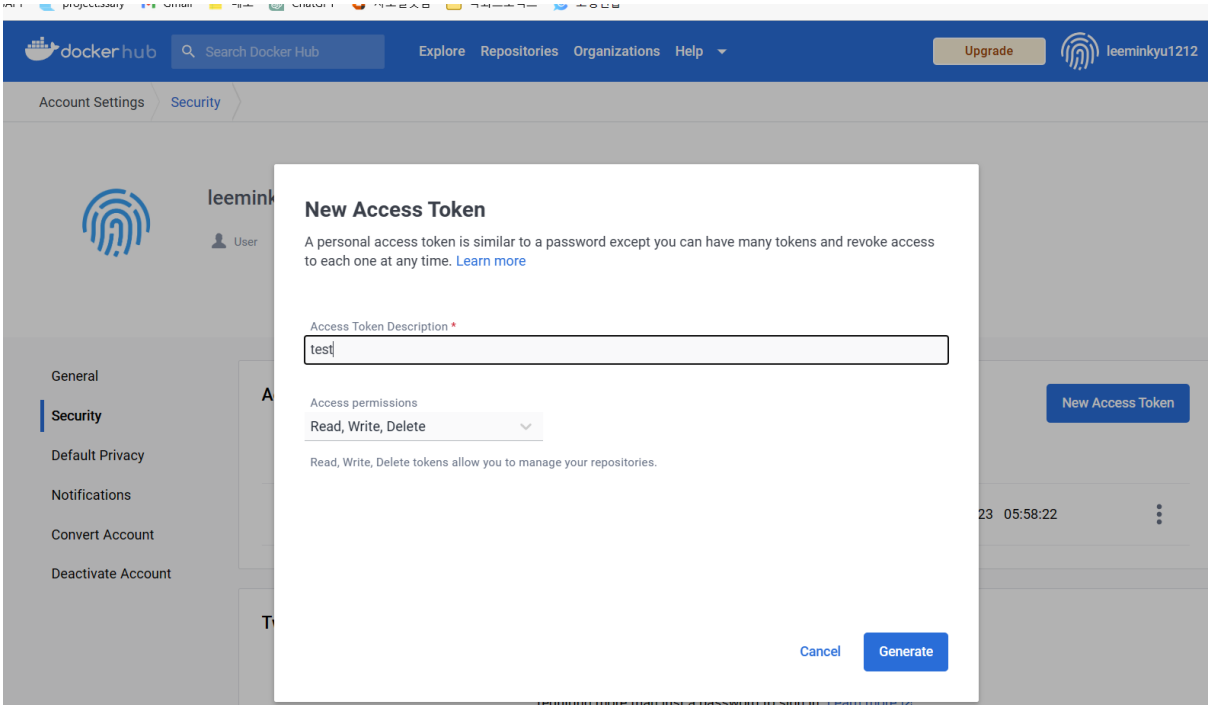
https://hub.docker.com/



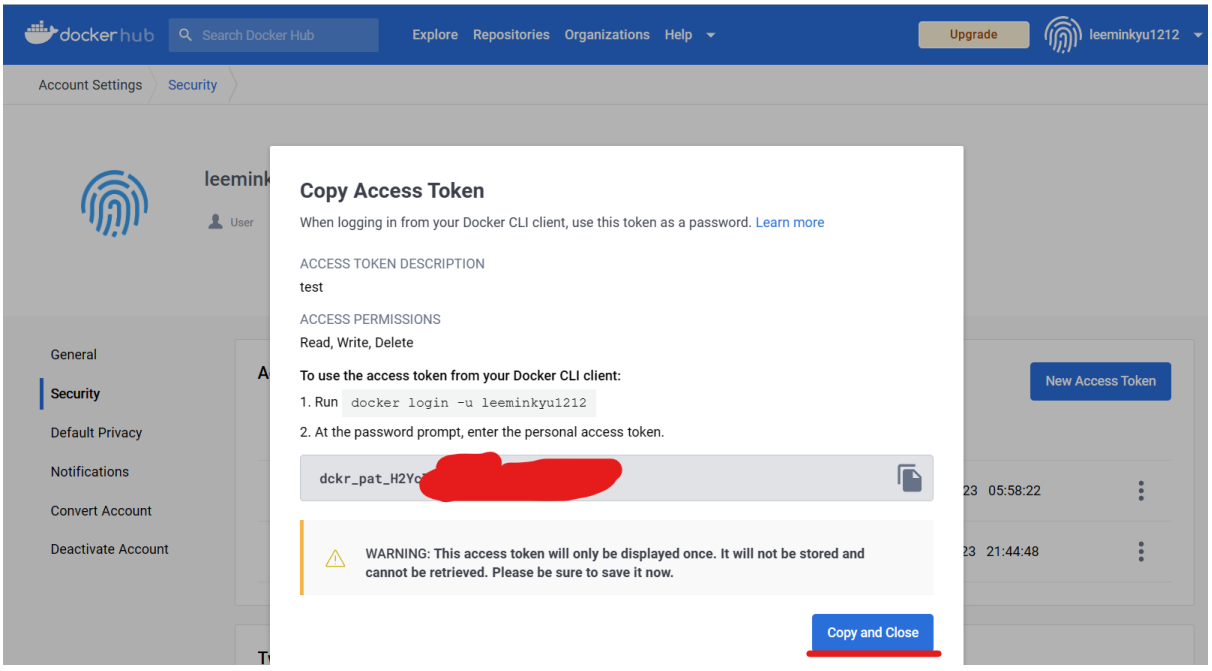
- 도커 허브 홈페이지 접속



우측 상단의 계정명 클릭 → Account Settings → Security → New Access Token



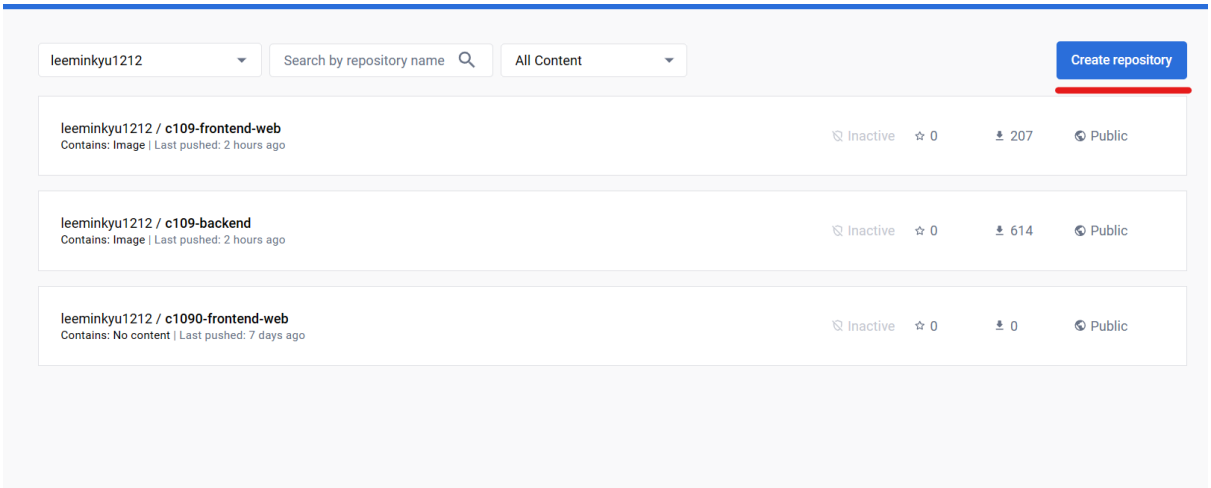
- Access Token 이름 지정
  - Access 권한 지정
  - 이후, **Generate** 버튼 클릭

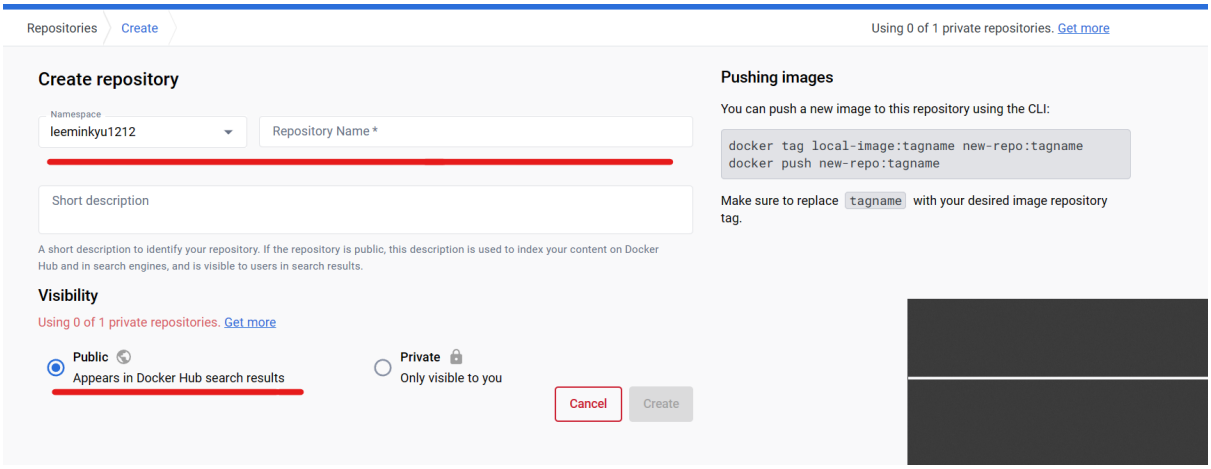


- 생성된 Access Token 복사 및 저장
  - 이후, **Copy and Close** 클릭

- **[주의] Access Token은 단 한 번만 공개되니 반드시 저장바람.** 실수로 지웠다면 삭제 후 재발급

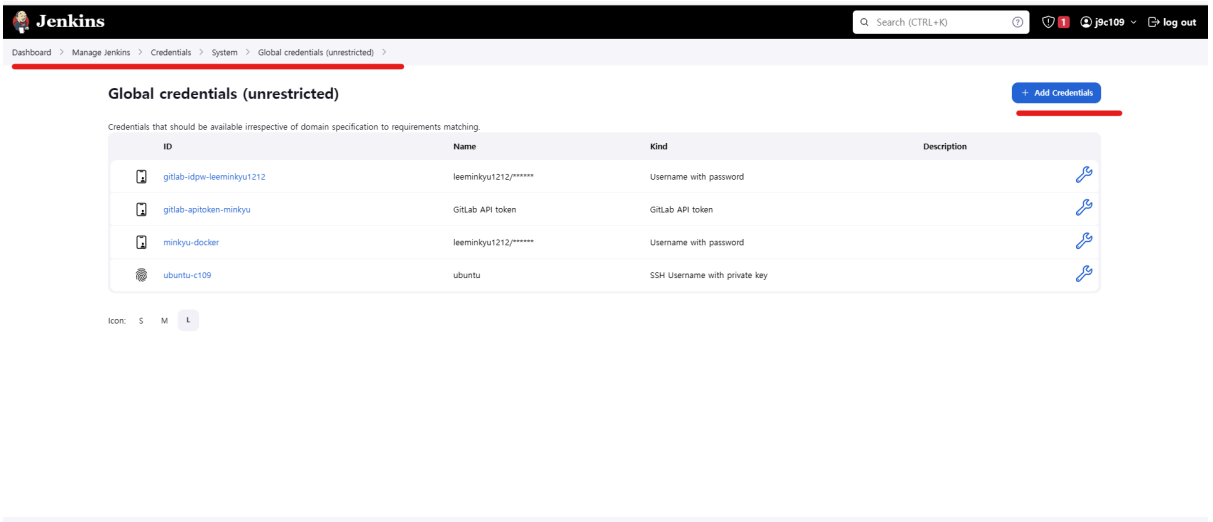
Docker Hub repository 생성





- Repositories → Create repostory 클릭
- Repository 이름 지정
  - Public 지정, Private는 계정당 한 개만 가능
  - Create 클릭

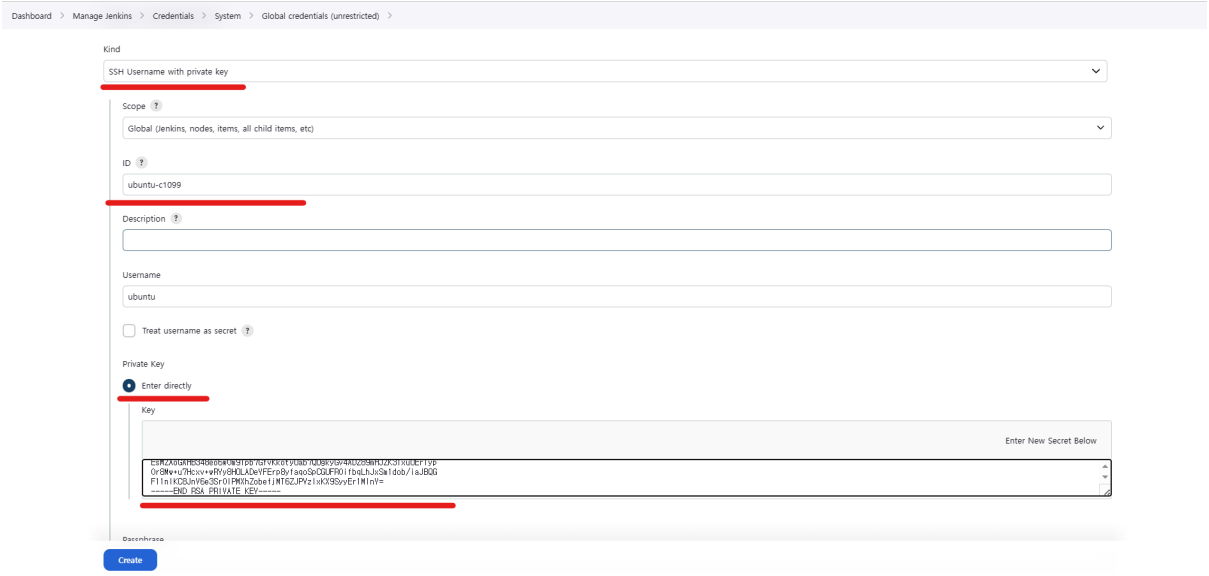
Docker Hub Credential 추가



- Dashboard → Manage Jenkins → Credentials System → Global credentials → Add credentials
- Credential 정보 입력
  - Kind: Username with password
  - Username: DockerHub에서 사용하는 계정 아이디 입력
  - Password: DockerHub에서 사용하는 Access Token 입력
  - ID: Jenkins 내부에서 사용하는 Credential 별칭 입력
  - Create

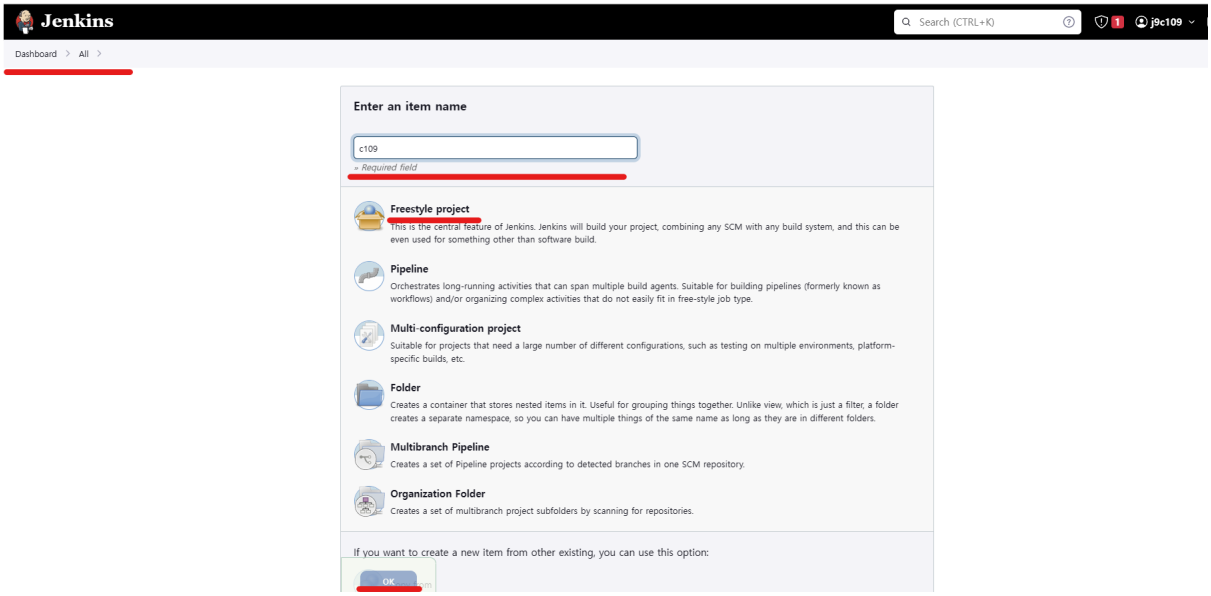
Ubuntu Credential 추가

필요 플러그인 : SSH Agent

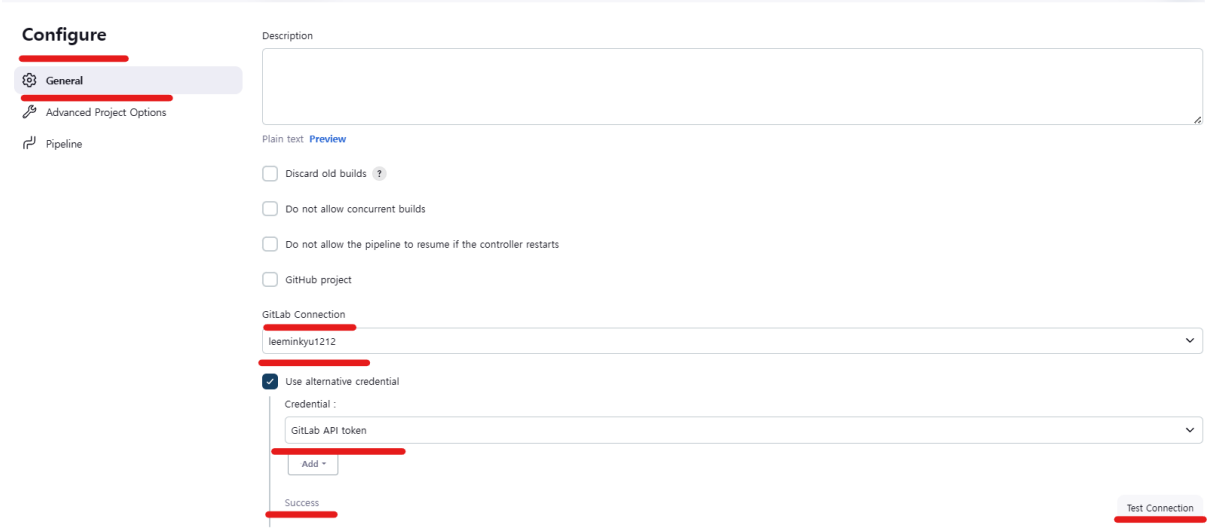


- Dashboard → Manage Jenkins → Credentials System → Global credentials → Add credentials
  - Kind - SSH Username with private key
  - ID: jenkins에서 Credential에 지정할 별칭 입력
  - Username: SSH 원격 서버 호스트에서 사용하는 계정명 입력
  - Enter directly 체크 → AWS 접속 pem키의 내용을 메모장으로 읽어 복사 후 Key에 붙여넣는다.

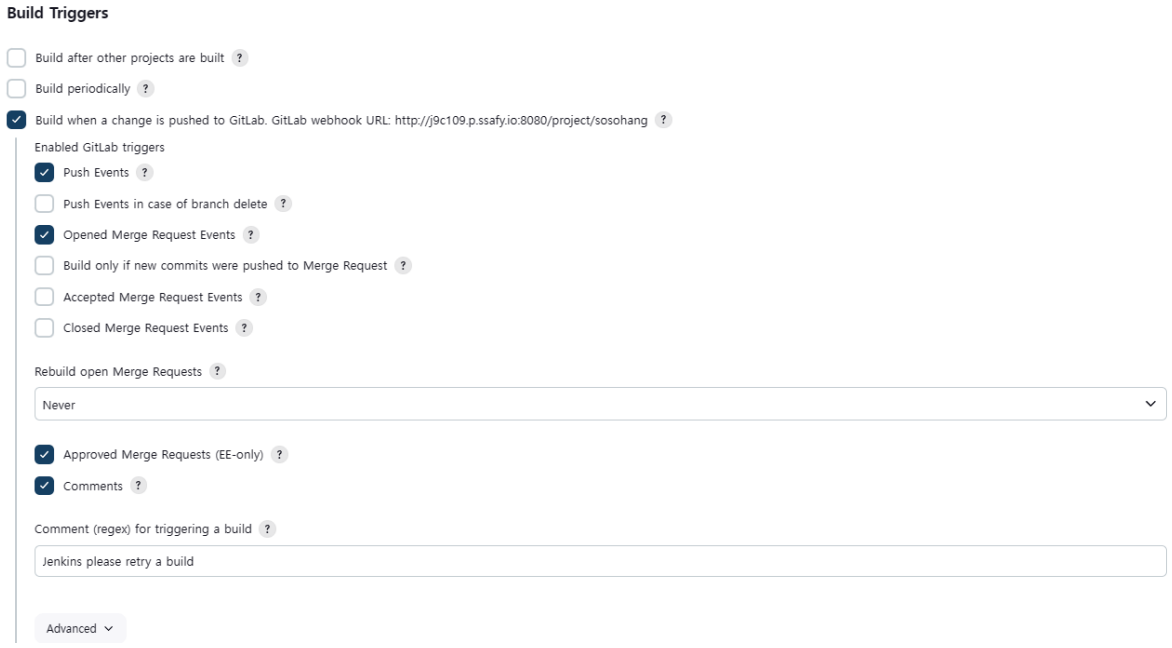
아이템 추가



- New Item→ pipeline 클릭 →OK



- Configure → General → Gitlab Connection



Build triggers의 Build when a change is pushed to GitLab 에 체크

Pipeline script 작성 예시

```
pipeline {
    agent any
    tools {nodejs "nodejs"} // Node.js 툴을 사용한다는 것을 명시

    // 환경변수 설정
    environment {
        imageName1 = 'leeminkyu1212/c109-backend'
        imageName2 = 'leeminkyu1212/c109-frontend-web'
        registryCredential = 'minkyu-docker'
        dockerImage = ''

        releaseServerAccount = 'ubuntu'
        releaseServerUri = 'j9c109.p.ssafy.io'
        releasePort1 = '8081'
        releasePort2 = '3000'
    }

    stages {

        // Git에서 코드를 Clone하는 단계
        stage('Git Clone') {
            steps {
                git branch: 'develop',
                    credentialsId: 'gitlab-idpw-leeminkyu1212',
                    url: 'https://lab.ssafy.com/s09-fintech-finance-sub2/S09P22C109'
            }
        }

        // JAR 파일과 npm 빌드를 수행하는 단계
        stage('Jar Build') {
            steps {
                dir('back-end-app') {
                    sh 'cp /var/jenkins_home/.backenv ./.env'
                    sh 'chmod 777 ./.env'
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar'
                }
                dir('front-end-web') {
                    sh 'cp /var/jenkins_home/.frontwebenv ./.env'
                    sh 'npm i'
                    sh 'npm run build'
                }
            }
        }

        // Docker 이미지를 빌드하고 DockerHub에 푸시하는 단계
        stage('Build & DockerHub Push') {
            steps {
                dir('back-end-app') {
                    script {
```

```
                        docker.withRegistry('', registryCredential) {
                            sh 'docker buildx create --use --name mybuilder'
                            sh "docker buildx build --platform linux/amd64,linux/arm64
-t $imageName1:$BUILD_NUMBER --push ."
                            sh "docker buildx build --platform linux/amd64,linux/arm64
-t $imageName1:latest --push ."
                        }
                    }
                }
                dir('front-end-web') {
                    script {
                        docker.withRegistry('', registryCredential) {
                            sh 'docker buildx create --use --name mybuilder'
                            sh "docker buildx build --platform linux/amd64,linux/arm64
-t $imageName2:$BUILD_NUMBER --push ."
                            sh "docker buildx build --platform linux/amd64,linux/arm64
-t $imageName2:latest --push ."
                        }
                    }
                }
            }
        }

        // 기존에 실행 중인 서비스를 중지하는 단계
        stage('Before Service Stop') {
            steps {
                sshagent(credentials: ['ubuntu-c109']) {
                    sh '''
                    if test "`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$r
eleaseServerUri "docker ps -aq --filter ancestor=$imageName1:latest"`"; then
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "docker stop $(docker ps -aq --filter ancestor=$imageName1:latest)"
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "docker rm -f $(docker ps -aq --filter ancestor=$imageName1:latest)"
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "docker rmi $imageName1:latest"
                    fi
                    if test "`ssh -o StrictHostKeyChecking=no $releaseServerAccount@$r
eleaseServerUri "docker ps -aq --filter ancestor=$imageName2:latest"`"; then
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "docker stop $(docker ps -aq --filter ancestor=$imageName2:latest)"
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "docker rm -f $(docker ps -aq --filter ancestor=$imageName2:latest)"
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "docker rmi $imageName2:latest"
                    fi
                    '''
                }
            }
        }

        // 서비스 서버에서 Docker 이미지를 Pull하는 단계
        stage('DockerHub Pull') {
            steps {
                sshagent(credentials: ['ubuntu-c109']) {
                    sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$release
```

```
ServerUri 'sudo docker pull $imageName1:latest'"
                }
                sshagent(credentials: ['ubuntu-c109']) {
                    sh "ssh -o StrictHostKeyChecking=no $releaseServerAccount@$release
ServerUri 'sudo docker pull $imageName2:latest'"
                }
            }
        }

        // 새 Docker 이미지로 서비스를 시작하는 단계
        stage('Service Start') {
            steps {
                sshagent(credentials: ['ubuntu-c109']) {
                    sh '''
                        ssh -o StrictHostKeyChecking=no $releaseServerAccount@$release
ServerUri "sudo docker run -i -e TZ=Asia/Seoul -e "SPRING_PROFILES_ACTIVE=prod" -v /ho
me/ubuntu/backup1:/backup1 --name sosohang -p $releasePort1:$releasePort1 -d $imageNam
e1:latest"
                    '''
                    sh '''
                    ssh -o StrictHostKeyChecking=no $releaseServerAccount@$releaseServ
erUri "sudo docker run -i -e TZ=Asia/Seoul -v /home/ubuntu/backup2:/backup2 --name sos
ohang-frontend -p $releasePort2:80 -d $imageName2:latest"
                    '''
                }
            }
        }

        // 서비스가 정상적으로 실행되는지 체크하는 단계
        stage('Service Check') {
            steps {
                sshagent(credentials: ['ubuntu-c109']) {
                    sh '''
                        #!/bin/bash
                        sleep 60
                        for retry_count in $(seq 20)
                        do
                          if curl -sf "http://j9c109.p.ssafy.io:8081/api/v1/test" > /d
ev/null
                          then
                            curl -d '{"text": "성공"}' -H "Content-Type: application/js
on" -X POST https://meeting.ssafy.com/hooks/e8bq89kszjg5ikatzmuqt9dazc
                            break
                          fi

                          if [ $retry_count -eq 20 ]
                          then
                            curl -d '{"text": "실패"}' -H "Content-Type: application/js
on" -X POST https://meeting.ssafy.com/hooks/e8bq89kszjg5ikatzmuqt9dazc
                            exit 1
                          fi

                          echo "The server is not alive yet. Retry health check in 5 s
econds..."
                          sleep 5
                        done
```

```
                    '''
                }
            }
        }
    }
}
```

Backend Dockerfile

```
# 기본 이미지로 OpenJDK 11을 사용
FROM openjdk:11-jre-slim
WORKDIR /app
COPY .env /app/.env
# JAR 파일을 컨테이너에 복사

COPY build/libs/back-end-app-0.0.1-SNAPSHOT.jar /app/back-end-app-0.0.1-SNAPSHOT.jar

# 외부와 통신할 포트 지정
EXPOSE 8080

# 컨테이너가 시작될 때 실행할 명령 지정
CMD ["java", "-jar", "/app/back-end-app-0.0.1-SNAPSHOT.jar"]
```

Frontend Dockerfile

```
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 3000 포트 오픈
EXPOSE 3000
EXPOSE 80
EXPOSE 443
```

```
# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

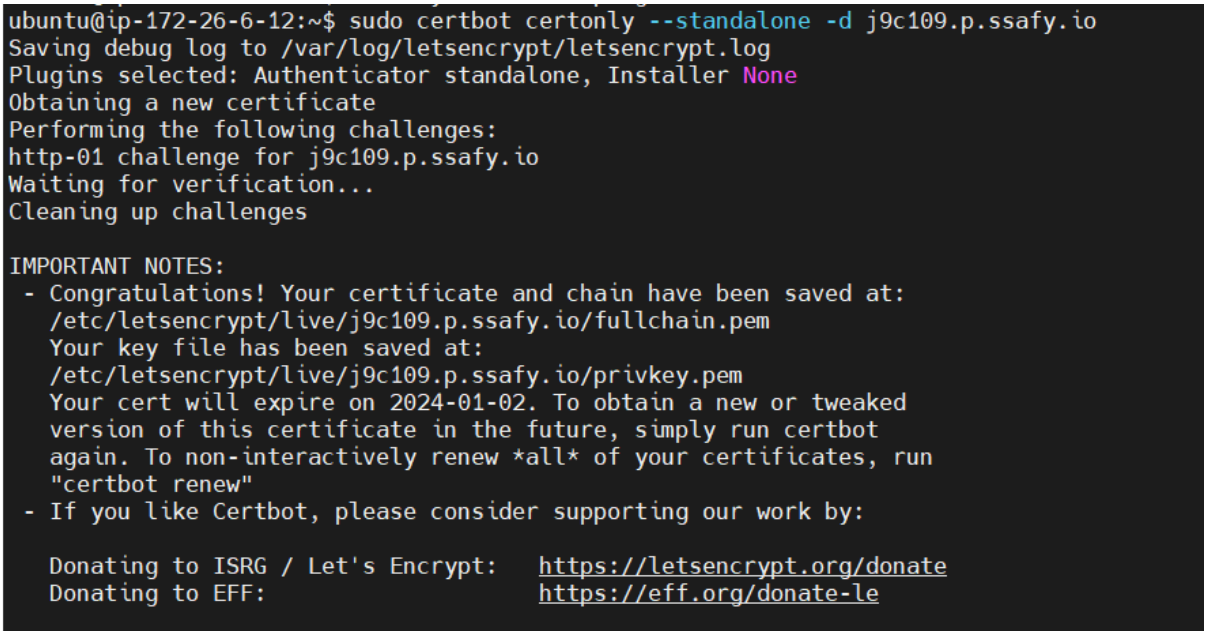
Nginx (Docker)

Let’s encrypt 로 SSL 인증서 발급받기

```
sudo apt install certbot
```

- certbot 클라이언트를 이용해 발급받기 위해 설치해 준다.

```
sudo certbot certonly --standalone -d j9c109.p.ssafy.io
```



- 명령어 실행후 certbot은 도메인 소유권 검증 절차를 시작하고, 완료되면 인증서가 발그급이 된다.

▼ 명령어 상세 설명

1. `sudo`: 관리자 권한으로 명령을 실행한다. Certbot을 실행하고 인증서를 설치하는 데는 루트 권한이 필요하다.

2. `certbot`: Let's Encrypt 클라이언트 프로그램인 Certbot을 호출한다.
3. `certonly`: 인증서만 발급하고 설치는 수동으로 진행하겠다는 옵션이다.
4. `-standalone`: Certbot의 내장 웹 서버를 사용하여 도메인 소유권을 검증한다. 이 옵션을 사용할 때는 80 포트와 443 포트가 열려 있어야 하며, 다른 웹 서버가 실행 중이면 종료해야 한다.
5. `d j9c109.p.ssafy.io`: 인증서를 발급받고자 하는 도메인 이름을 지정한다. `d` 옵션 뒤에 도메인을 입력한다.

```
docker pull nginx
```

- nginx를 도커로 띄우기 위해 pull 받아온다.

```
# nginx.conf
# 이벤트 관련 설정
events{
    # 동시에 처리할 수 있는 연결 수 설정
    worker_connections 1024;
}

# HTTP 관련 설정 시작
http{
    # 서버 설정 시작
    server {
        # 80번 포트(HTTP)에서 리스닝 설정
        listen 80;
        # 서버 이름 설정
        server_name j9c109.p.ssafy.io;

        # SSL(HTTPS) 관련 설정
        # 443번 포트(HTTPS)에서 리스닝 설정
        listen 443 ssl;
        # SSL 인증서 위치
        ssl_certificate /etc/nginx/fullchain.pem;
        # SSL 키 파일 위치
        ssl_certificate_key /etc/nginx/privkey.pem;

        # /jenkins/ 경로로 시작하는 요청에 대한 리버스 프록시 설정
        location /jenkins/ {
            # Jenkins 서버로 요청 전달
            proxy_pass http://j9c109.p.ssafy.io:8080;
            # 요청 헤더 설정
```

```
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        # /api/ 경로로 시작하는 요청에 대한 리버스 프록시 설정
        location /api/ {
            # API 서버로 요청 전달
            proxy_pass http://j9c109.p.ssafy.io:8081;
            # 요청 헤더 설정
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }

        # 나머지 모든 경로에 대한 리버스 프록시 설정
        location / {
            # 웹 애플리케이션 서버로 요청 전달
            proxy_pass http://j9c109.p.ssafy.io:3000;
            # 요청 헤더 설정
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        }
    } # 서버 설정 종료
} # HTTP 관련 설정 종료
```

- 리버스 프록시 설정을 통해 클라이언트는 단일 도메인 및 포트로 접근하게 되지만 ,실제로는 내부에서 여러 서비스와 포트로 요청이 전달되는 구조이다.
- /api 엔드포인트로 들어오는 요청은 8081 포트로 리다이렉트 되고,
  /jenkins 엔드포인트로 들어오는 요청은 8080 포트로 리다이렉트 된다.
- 나머지 경로로 들어오는 요청은 모두 3000번 포트로 매핑된다.

```
mkdir nginx
cd nginx
touch Dockerfile
```

- 리버스 프록시용 nginx 기반 이미지를 만들기 위해 도커파일을 만들어 준다.

```
sudo cp /etc/letsencrypt/live/j9c109.p.ssafy.io/fullchain.pem ./nginx/
sudo cp /etc/letsencrypt/live/j9c109.p.ssafy.io/privkey.pem ./nginx/
sudo chmod 755 fullchain.pem privkey.pem
```

- 도커파일이 접근할수 있도록 해당 디렉토리에 복사해주고 ,권한을 허용해준다.

```
# ./nginx/Dockerfile

# 베이스 이미지 설정
FROM nginx:latest

# nginx.conf 파일을 컨테이너의 /etc/nginx/nginx.conf 로 복사
COPY nginx.conf /etc/nginx/nginx.conf

# SSL 인증서 복사 (로컬에 있는 인증서 파일을 사용한다고 가정)
COPY fullchain.pem /etc/nginx/fullchain.pem
COPY privkey.pem /etc/nginx/privkey.pem

# 80과 443 포트 열기
EXPOSE 80 443

# NGINX 실행
CMD ["nginx", "-g", "daemon off;"]
```

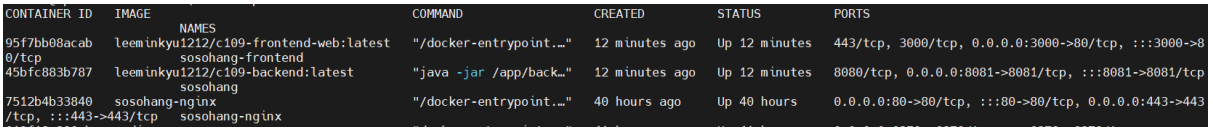
- Dockerfile 예시

```
docker build -t sosohang-nginx .
```

- 상단 Dockerfile기반 도커 이미지를 생성한다.

```
docker run --name nginx-container -d -p 80:80 -p 443:443 sosohang-nginx
```

- 해당 이미지를 기반으로 80포트와 443 포트를 바인딩해주는 컨테이너를 생성한다.



```
CONTAINER ID   IMAGE                                      COMMAND                  CREATED          STATUS          PORTS                                                                                  NAMES
95f7bb08acab   leeminkyu1212/c109-frontend-web:latest     "/docker-entrypoint.…"   12 minutes ago   Up 12 minutes   443/tcp, 3000/tcp, 0.0.0.0:3000->80/tcp, :::3000->80/tcp                               sosohang-frontend
45bfc883b787   leeminkyu1212/c109-backend:latest          "java -jar /app/back…"   12 minutes ago   Up 12 minutes   8080/tcp, 0.0.0.0:8081->8081/tcp, :::8081->8081/tcp                                    sosohang
7512b4b33840   sosohang-nginx                             "/docker-entrypoint.…"   40 hours ago     Up 40 hours     0.0.0.0:80->80/tcp, :::80->80/tcp, 0.0.0.0:443->443/tcp, :::443->443/tcp   sosohang-nginx
```

- 성공 확인

MariaDB (Docker)

```
docker pull mariadb
```

mariadb를 도커로 띄우기 위해 이미지를 pull 받아온다.

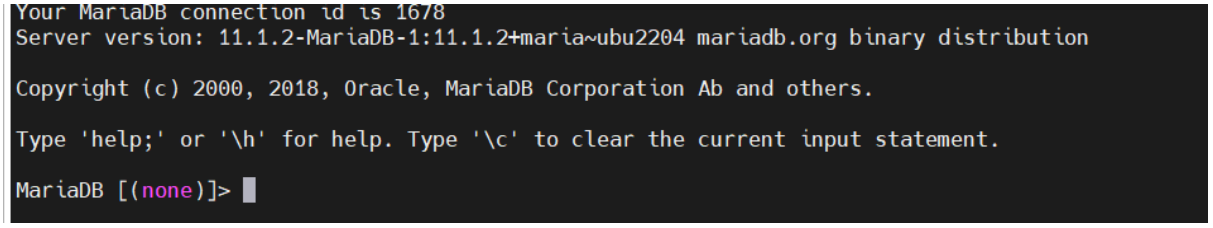
```
docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=비밀번호 -v /var/lib/mysql:/var/lib/my
sql --name mariadb mariadb
```

▼ 명령어 상세 설명

1. `docker run`: 새로운 컨테이너를 실행하기 위한 Docker 명령어다.
2. `d`: 컨테이너를 백그라운드 모드로 실행한다.
3. `p 3306:3306`: 호스트의 3306 포트와 컨테이너의 3306 포트를 연결한다. `p` 옵션을 사용하여 호스트와 컨테이너 간의 포트 포워딩을 설정한다.
4. `e MYSQL_ROOT_PASSWORD=비밀번호`: 컨테이너 환경 변수를 설정한다. 여기서는 MariaDB의 `root` 사용자의 비밀번호를 설정한다.
5. `v /var/lib/mysql:/var/lib/mysql`: 호스트의 `/var/lib/mysql` 디렉토리와 컨테이너의 `/var/lib/mysql` 디렉토리를 연결한다. `v` 옵션을 사용하여 볼륨을 마운트한다. 이렇게 하면 컨테이너가 종료되어도 데이터가 손실되지 않는다.
6. `-name mariadb`: 실행하는 컨테이너에 `mariadb`라는 이름을 부여한다.
7. `mariadb`: 실행할 이미지의 이름이다. 여기서는 Docker Hub에서 MariaDB 이미지를 가져와 실행한다.

```
docker exec -it mariadb /bin/bash

mariadb
```



MariaDB 외부접속 설정

- 이 설정을 하지 않으면 localhost 환경에서만 MariaDB 접속이 가능하므로 모든 외부 아이피에 대해 개방 설정을 진행한다.

```
sudo sed -i 's/127.0.0.1/0.0.0.0/g' /etc/mysql/mariadb.conf.d/50-server.cnf
```

MariaDB root 비밀번호 변경

- **작은 따옴표 사이에 변경할 비밀번호를 입력한다.**

```
sudo mysql -u root -e "set password for 'root'@'localhost' = password('비밀번호'); flush privileges;"
```

MariaDB root 계정 외부 접속 설정

- MariaDB 내의 root 계정은 기본적으로 127.0.0.1에 대한 아이피만 접속이 가능하다. 원격 접속을 허용하기 위해 다음 명령어를 입력한다.
  - **root 비밀번호 입력 칸이 두 개임에 주의할 것!**

```
sudo mysql -u root -p 비밀번호 -e "grant all privileges on *.* to 'root'@'%' identified by '비밀번호'; flush privileges;"
```

MariaDB 서버 시간을 한국 표준시(UTC+9)로 변경

```
sudo sed -i '$s/$/\n\n[mysqld]\ndefault-time-zone='+9:00'/g' /etc/mysql/mariadb.conf.d/50-server.cnf
```

데이터베이스 접속

- root 혹은 다른 계정정보 입력 후 비밀번호 입력
  - `-u` : 계정 이름
  - `-p` : 비밀번호 입력을 위한 옵션, 명령어 입력 후 비밀번호 별도 기입

```
sudo mysql -u MariaDB계정이름 -p
```

데이터베이스 생성

```
create database 데이터베이스이름;
```

생성한 데이터베이스 사용

```
use 데이터베이스이름;
```

MariaDB 사용자 목록 조회

```
use mysql;
select host, user, password from user;
```

사용자 계정 생성

- `'계정이름'@'localhost'` : 로컬 환경에서만 계정 사용 가능
- `'계정이름'@'%'` : 모든 아이피에서 계정 사용 가능

```
create user '계정이름'@'%' identified by '비밀번호';
```

사용자 계정 삭제

```
drop user 계정이름@아이피주소;
```

사용자에게 데이터베이스 사용 권한 부여

```
grant all privileges on 데이터베이스이름.* to '계정이름'@'%';
```

변경한 환경 설정 반영

```
flush privileges;
```

Redis (Docker)

```
sudo docker pull redis
```

- docker의 pull 명령을 통해 latest 버전의 redis 이미지를 다운로드 한다

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

- redis 컨테이너 명령어를 통해 서비스를 띄운다

▼ 명령어 상세 설명

`run` : 이미지를 가지고 컨테이너를 만들고 컨테이너 실행

`-d` : 컨테이너를 만들고 백그라운드에 계속 구동하게 하는 옵션 (데몬)

`-p` : host:container 포트 연결 (6379)

`--name redis` : 컨테이너 이름 지정 (미지정시 무작위로 이름 지정)

`redis` : dockerhub의 redis 이미지 사용

```
docker exec -it redis /bin/bash
```

- redis 컨테이너 내부에 접속한다.

```
sed -i 's/127.0.0.1/0.0.0.0/g' /etc/redis/redis.conf
```

- 코드 변경을 통해 외부 접속도 허용하게 된다.

Back-end API 서버 구축

build.gradle

```
// 플러그인 설정 부분
plugins {
  // Java 프로젝트를 위한 플러그인
  id 'java'
  // Spring Boot 프로젝트를 위한 플러그인
  id 'org.springframework.boot' version '2.7.15'
  // Spring 의존성 관리를 위한 플러그인
  id 'io.spring.dependency-management' version '1.0.15.RELEASE'
}

// 프로젝트의 그룹 및 버전 정보 설정
group = 'project.app.c109'
version = '1.0.0'

// 사용할 Java 버전 설정
java {
  sourceCompatibility = '11'
}

// 컴파일 옵션 설정
configurations {
  compileOnly {
    // 컴파일 시에만 필요한 의존성 설정 (예: Lombok)
    extendsFrom annotationProcessor
  }
}
```

```
// 의존성을 다운로드할 저장소 설정 (Maven Central 저장소 사용)
repositories {
  mavenCentral()
}

// 프로젝트에서 사용할 의존성 목록
dependencies {
implementation 'org.springframework.boot:spring-boot-starter-actuator'
  implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
  implementation 'org.springframework.boot:spring-boot-starter-security'
  implementation 'org.springframework.boot:spring-boot-starter-web'
  implementation 'org.springframework.boot:spring-boot-starter-validation'
  compileOnly 'org.projectlombok:lombok'
  runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
  annotationProcessor 'org.projectlombok:lombok'
  testImplementation 'org.springframework.boot:spring-boot-starter-test'
  testImplementation 'org.springframework.security:spring-security-test'

  //Zxing
  implementation 'com.google.zxing:core:3.4.1'
  implementation 'com.google.zxing:javase:3.4.1'

  //SWAGGER
  implementation 'org.springdoc:springdoc-openapi-ui:1.6.12'

  // JWT
  implementation 'io.jsonwebtoken:jjwt-api:0.11.5'
  implementation 'io.jsonwebtoken:jjwt-impl:0.11.5'
  implementation 'io.jsonwebtoken:jjwt-jackson:0.11.5'

  // Redis
  implementation 'org.springframework.boot:spring-boot-starter-data-redis'

  // Logging
  implementation 'org.springframework.boot:spring-boot-starter-logging'

  // AWS
  implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE'
  implementation 'software.amazon.awssdk:s3:2.17.66'

    // env 실행
  implementation 'io.github.cdimascio:java-dotenv:5.2.2'
}

// 테스트 실행 시 JUnit 플랫폼 사용 설정
tasks.named('test') {
  useJUnitPlatform()
}
```

▼ 코드 상세 설명

- **플러그인 설정**

    - `java`: Java 기반 프로젝트임을 명시한다.
    - `org.springframework.boot`: Spring Boot 프로젝트임을 명시하며, 특정 버전을 사용한다.
    - `io.spring.dependency-management`: Spring의 의존성 관리를 위한 플러그인이다.
- **그룹 및 버전 설정**
    - 프로젝트의 그룹 ID와 버전을 설정한다.
- **자바 버전 설정**
    - Java 11 버전을 사용함을 명시한다.
- **의존성 구성 설정**
    - `compileOnly`는 컴파일 시에만 필요한 의존성을 의미하며, 여기서는 annotationProcessor를 확장한다. Lombok과 같은 코드 생성 도구들이 여기에 해당한다.
- **저장소 설정**
    - 의존성을 다운로드 받기 위한 중앙 저장소로 mavenCentral()을 사용한다.
- **의존성 추가**
    - Spring Boot 관련 스타터 의존성: actuator, data-jpa, security, web, validation 등을 추가한다.
    - Lombok: 코드를 간결하게 작성할 수 있게 도와주는 라이브러리다.
    - mariadb-java-client: MariaDB 데이터베이스와의 연결을 위한 JDBC 드라이버다.
    - Zxing: 바코드 및 QR 코드 생성/해독 라이브러리다.
    - Springdoc: OpenAPI와 Swagger UI를 위한 의존성이다.
    - JWT: JSON Web Token을 다루기 위한 라이브러리다.
    - Redis: Spring Data Redis 스타터를 추가한다.
    - Logging: 로깅을 위한 스타터다.
    - AWS: Amazon S3와 연동하기 위한 의존성들이다.
    - java-dotenv: 환경 변수를 로드하기 위한 라이브러리다.
- **테스트 설정**
    - JUnit 플랫폼을 사용하여 테스트를 실행한다.

Back-end Dockerfile

```
# 기본 이미지로 OpenJDK 11을 사용
FROM openjdk:11-jre-slim
WORKDIR /app
COPY .env /app/.env
# JAR 파일을 컨테이너에 복사

COPY build/libs/back-end-app-0.0.1-SNAPSHOT.jar /app/back-end-app-0.0.1-SNAPSHOT.jar

# 외부와 통신할 포트 지정
EXPOSE 8081

# 컨테이너가 시작될 때 실행할 명령 지정
CMD ["java", "-jar", "/app/back-end-app-0.0.1-SNAPSHOT.jar"]
```

Front-end 웹 구축

dockerfile

```
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 3000 포트 오픈
EXPOSE 3000
EXPOSE 80
EXPOSE 443
```

```
# container 실행 시 자동으로 실행할 command. nginx 시작함
CMD ["nginx", "-g", "daemon off;"]
```

package.json

```
{
  "name": "front-end-web",
  "version": "0.1.0",
  "private": true,
  "dependencies": {
    "@babel/plugin-proposal-private-property-in-object": "^7.21.11",
    "@testing-library/jest-dom": "^5.17.0",
    "@testing-library/react": "^13.4.0",
    "@testing-library/user-event": "^13.5.0",
    "aws-sdk": "^2.1465.0",
    "axios": "^1.5.0",
    "bootstrap": "^5.3.2",
    "chart.js": "^4.4.0",
    "dotenv": "^16.3.1",
    "http-proxy-middleware": "^2.0.6",
    "js-cookie": "^3.0.5",
    "react": "^18.2.0",
    "react-awesome-reveal": "^4.2.5",
    "react-bootstrap": "^2.8.0",
    "react-chartjs-2": "^5.2.0",
    "react-daum-postcode": "^3.1.3",
    "react-dom": "^18.2.0",
    "react-router-dom": "^6.15.0",
    "react-scripts": "5.0.1",
    "styled-components": "^6.0.8",
    "web-vitals": "^2.1.4"
  },
  "scripts": {
    "start": "react-scripts start",
    "build": "react-scripts build",
    "test": "react-scripts test",
    "eject": "react-scripts eject"
  },
  "eslintConfig": {
    "extends": [
      "react-app",
      "react-app/jest"
    ]
  },
  "browserslist": {
    "production": [
      ">0.2%",
      "not dead",
      "not op_mini all"
    ],
    "development": [
```

```
      "last 1 chrome version",
      "last 1 firefox version",
      "last 1 safari version"
    ]
  }
}
```

소소행 어플리케이션 구축

expo 플랫폼을 통한 앱 빌드 과정

````
1. **expo-cli 라이브러리 설치**
   - **디렉토리 위치 & 파일명**: 전역으로 설치하기 때문에 특정 디렉토리 위치나 파일명은 없다.
   ```bash
   npm install -g expo-cli
   ```

2. **expo 로그인**
   - 이 과정은 expo의 웹 서비스에 로그인해서 자신의 프로젝트를 관리할 수 있게 한다.
   ```bash
   expo login
   ```

3. **eas-cli 라이브러리 설치**
   - **디렉토리 위치 & 파일명**: 전역으로 설치되기 때문에 특정 디렉토리나 파일명은 없다.
   ```bash
   npm install -g eas-cli
   ```

4. **eas.json 설정**
   - **디렉토리 위치 & 파일명**: 프로젝트의 루트 디렉토리에 `eas.json` 파일을 생성하거나 수정한다.
```json
   {
     "cli": {
       "version": ">=5.2.0"
     },
     "build": {
       "preview": {
         "android": {
           "buildType": "apk"
         }
       },
       "preview2": {
         "android": {
           "gradleCommand": ":app:assembleRelease"
````
````

```
        }
      },
      "preview3": {
        "developmentClient": true
      },
      "production": {}
    }
  }
  ```

5. **앱 빌드 명령어 실행**
   - **디렉토리 위치 & 파일명**: 프로젝트의 루트 디렉토리에서 이 명령어를 실행한다.
   ```bash
   eas build -p android --profile preview
   ```

6. **app.config.js 수정**
   - **디렉토리 위치 & 파일명**: 프로젝트의 루트 디렉토리에 있는 `app.config.js` 파일에 추가한다.
   ```javascript
   {
      ...
     {
     "expo": {
       "extra": {
         "eas": {
           "projectId": "프로젝트 아이디"
         }
       }
     }
      ...
   }
   }
   ```

7. **링크로부터 .apk 파일 다운로드**
   - 빌드가 완료되면 나타나는 링크를 통해 `.apk` 파일을 다운로드 받을 수 있다. 이 링크는 Expo의 웹 페이지에서 제공된다.

소소행 소비자용 앱 package.json

```
{
  "name": "front-end-customer",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "tunnel": "expo start --tunnel",
    "android": "expo start --android",
    "ios": "expo start --ios",
    "web": "expo start --web"
  },
  "dependencies": {
```

```
    "@actbase/react-daum-postcode": "^1.0.4",
    "@expo/webpack-config": "^19.0.0",
    "@react-native-async-storage/async-storage": "1.18.2",
    "@react-native-clipboard/clipboard": "^1.12.1",
    "@react-navigation/native": "^6.1.7",
    "@react-navigation/stack": "^6.3.17",
    "@rneui/themed": "^4.0.0-rc.8",
    "aws-sdk": "^2.1470.0",
    "axios": "^1.5.0",
    "buffer": "^6.0.3",
    "crypto-js": "^4.1.1",
    "expo": "^49.0.0",
    "expo-camera": "~13.4.4",
    "expo-checkbox": "~2.4.0",
    "expo-contacts": "~12.2.0",
    "expo-file-system": "~15.4.4",
    "expo-image-manipulator": "~11.3.0",
    "expo-image-picker": "~14.3.2",
    "expo-location": "~16.1.0",
    "expo-status-bar": "~1.6.0",
    "iamport-react-native": "^2.0.6",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-native": "0.72.5",
    "react-native-aws3": "^0.0.9",
    "react-native-gesture-handler": "^2.13.1",
    "react-native-image-resizer": "^1.4.5",
    "react-native-maps": "1.7.1",
    "react-native-safe-area-context": "^4.7.2",
    "react-native-screens": "^3.25.0",
    "react-native-web": "~0.19.6",
    "react-native-webview": "^11.26.1"
  },
  "devDependencies": {
    "@babel/core": "^7.20.0",
    "react-native-dotenv": "^3.4.9"
  },
  "private": true
}
```

소소행 사장님 용 앱 package.json

```
{
  "name": "sosohang-owner",
  "version": "1.0.0",
  "main": "node_modules/expo/AppEntry.js",
  "scripts": {
    "start": "expo start",
    "tunnel": "expo start --tunnel",
    "android": "expo start --android",
```

```
    "ios": "expo start --ios",
    "web": "expo start --web"
  },
  "dependencies": {
    "@actbase/react-daum-postcode": "^1.0.4",
    "@expo/webpack-config": "^19.0.0",
    "@react-native-async-storage/async-storage": "1.18.2",
    "@react-native-clipboard/clipboard": "^1.12.1",
    "@react-navigation/native": "^6.1.7",
    "@react-navigation/stack": "^6.3.17",
    "@rneui/themed": "^4.0.0-rc.8",
    "axios": "^1.5.0",
    "expo": "^49.0.13",
    "expo-barcode-scanner": "~12.5.3",
    "expo-camera": "~13.4.4",
    "expo-checkbox": "~2.4.0",
    "expo-contacts": "~12.2.0",
    "expo-image-picker": "~14.3.2",
    "expo-location": "~16.1.0",
    "expo-status-bar": "~1.6.0",
    "react": "18.2.0",
    "react-dom": "18.2.0",
    "react-native": "0.72.5",
    "react-native-gesture-handler": "~2.12.0",
    "react-native-maps": "1.7.1",
    "react-native-safe-area-context": "4.6.3",
    "react-native-screens": "~3.22.0",
    "react-native-web": "~0.19.6",
    "react-native-webview": "13.2.2"
  },
  "devDependencies": {
    "@babel/core": "^7.20.0",
    "react-native-dotenv": "^3.4.9"
  },
  "private": true
}
```

---

2. 프로젝트 외부 서비스 정보 정리 문서

- front-end-customer

```
EXPO_GOOGLE_API_KEY = 지도 페이지에 필요한 Google Map API Key
EXPO_PG_USER_CODE = 아임포트 결제에 필요한 사용자 코드
EXPO_APP_AWS_ACCESS_KEY_ID= 소소티콘 생성에 필요한 AWS ACCESS KEY
EXPO_APP_AWS_SECRET_ACCESS_KEY= 소소티콘 생성에 필요한 Secret Access Key
EXPO_APP_AWS_REGION= 소소티콘 생성에 필요한 AWS Region
EXPO_APP_AWS_BUCKET_NAME= 소소티콘 생성에 필요한 Bucke Name
```

- front-end-web

```
REACT_APP_AWS_ACCESS_KEY_ID= 사장님 상점, 상품 이미지 등록을 위한 AWS Access Key
REACT_APP_AWS_SECRET_ACCESS_KEY= 사장님 상점, 상품 이미지 등록을 위한 AWS Secret Acces Key
REACT_APP_AWS_REGION= 사장님 상점, 상품 이미지 등록을 위한 AWS Region
REACT_APP_BUCKET_NAME= 사장님 상점, 상품 이미지 등록을 위한 AWS Bucket Name
REACT_APP_KAKAO_ACCESS_KEY= 사장님 상점, 상품 이미지 등록을 위한 카카오 Access Key
```

---

3. DB 덤프 파일 최신본

파일 첨부 : sosohang_database.sql

---

4. 시연 시나리오

1. 회원앱 : 정빈

“시연 시작하겠습니다. 어어 죄송합니다. 잠시만요? 누가 선물을 보냈네요?”

- 소소티콘 확인

“저도 선물해야할 친구가 있어서요.”

- 지도에 동네 찍고 카테고리 둘러보기
- 상점 한 곳 선택해서 2개 구매
- 결제 수단 : 삼성페이

2. 사장님앱 : 민규

“요즘 이 어플 많이들 쓰시네요. 덕분에 매출이 많이 올랐어요.”

- QR 분할 결제
- 스탬프 적립