포팅메뉴얼

Gitlab 소스 클론 이후 빌드 및 배포할 수 있도록 정리한 문서

1)사용한 JVM, 웹서버, WAS 제품 등의 종류와 설정값, 버전(IDE버전 포함) 기재

WAS : 톰캣

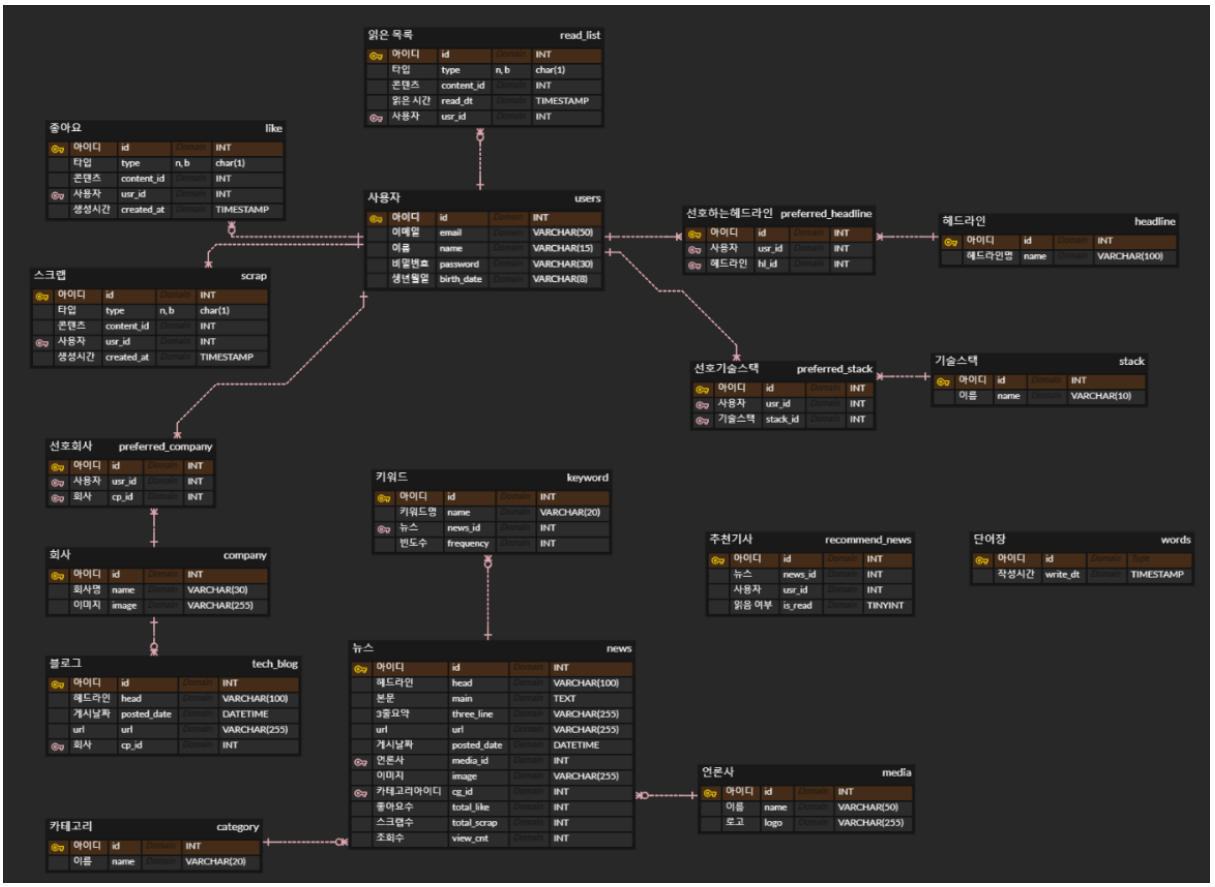
intelij : 2023.2 Ultimate

```
spring:
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://j9b202.p.ssafy.io:3350/newssumdb2?useUniCode=yes&characterEncoding=UTF-8&serverTimezone=Asia/Seoul
    username: newsum
    password: k3s2b202ssafy

  jpa:
    show-sql: true
    database: mysql
    properties:
      hibernate:
        format_sql: true
        dialect: org.hibernate.dialect.MySQL8Dialect

  security:
    user:
      name: admin
      password: admin

# JPA log
logging:
  level:
    org:
      hibernate:
        SQL: DEBUG
        type:
          descriptor:
            sql:
              BasicBinder: TRACE

# smtp
mail:
  smtp:
    port: 465
    socketFactory:
      port: 465
      class: javax.net.ssl.SSLSocketFactory
      fallback: false
    auth: true
    starttls:
      required: true
      enable: true

# admin 구글 계정
AdminMail:
  id:
  password:

# jwt
jwt:
```

```
  secret: d39gssddf6SsdahASDFFaesdGHJFDGjsdfsddfsdhSDS2HSZDasEsd48Os923459780012I4E3IdjSLOFJaseyH
  access:
    expiration: 1209600000 #  (1000L(ms -> s) * 60L(s -> m) * 60L(m -> h) * 24L(h -> 하루) * 14(2주))
    header: Authorization
    # expiration: 3600000 # 1시간(60분) (1000L(ms -> s) * 60L(s -> m) * 60L(m -> h))

  refresh:
    expiration: 1209600000 #  (1000L(ms -> s) * 60L(s -> m) * 60L(m -> h) * 24L(h -> 하루) * 14(2주))
    header: Authorization-refresh


server:
  port: 8811
  max-http-header-size: 2MB
```

2)DB 접속 정보 등 프로젝트(ERD)에 활용되는 주요 계정 및 프로퍼티가 정의된 파일 목록

Untitled



인프라

▼ SSL 인증 받기

https://www.vompressor.com/tls1/

1. Cerbot 설치를 snap을 통해

```
sudo snap install cerbot --classic
```

2. standalone 방식으로 인증서 발급
   - 밑에 명령문을 하기전에 cerbot의 경로에 가서 실행해야함
   - snap 폴더의 cerbot 폴더로 들어가서 밑에 실행
   - 80번 포트가 개방되어야함

     ```
     sudo ufw allow 80
     sudo ufw enable

     sudo ufw status -> 현재 allow 되어있는 포트 확인
     ```

   - cerbot이 인증서를 발급할때 도메인의 소유주임을 확인하기 위해 80번 포트를 사용하는 간이 웹서버를 가동한다. 이 방식으로 인증서를 발급하려면 80번 포트를 점유하는 서비스를 중지해야한다(예를들면 nginx 그래서 nginx 먼저 깔기 전에 ssl 인증 받기)

```
sudo cerbot certonly --standalone
```

```
dev@vultr:~$ sudo certbot certonly --standalone
[sudo] password for dev:   <root password>
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator standalone, Installer None
Enter email address (used for urgent renewal and security notices)
 (Enter 'c' to cancel): vompressor@gmail.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf. You must
agree in order to register with the ACME server. Do you agree?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y <- ACME 약관에 동의하는지 N선택시 진행불가

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Would you be willing, once your first certificate is successfully issued, to
share your email address with the Electronic Frontier Foundation, a founding
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: N <- 이메일을 통해 Let's Encrypt 프로젝트 정보를 받아볼지
Please enter in your domain name(s) (comma and/or space separated)
(Enter 'c' to cancel): vompressor.com www.vompressor.com  <- {1} 인증서를 발급할 도메인 입력
Requesting a certificate for vompressor.com and www.vompressor.com

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/vompressor.com/fullchain.pem <- {2} 발급된 인증서 경로
   Your key file has been saved at:
   /etc/letsencrypt/live/vompressor.com/privkey.pem <- {2} 발급된 인증서 경로
   Your certificate will expire on 2021-05-16. To obtain a new or
   tweaked version of this certificate in the future, simply run
   certbot again. To non-interactively renew *all* of your
   certificates, run "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

{2} → 인증서 경로와 인증서 key 경로를 알고 있어야 nginx 설정할 때 사용해야 함

3. 443 포트 열어주기

```
sudo ufw allow 443
sudo ufw enable
```

▼ Nginx

1. 서버의 패키지 목록 업데이트

```
sudo apt update
```

2. nginx 설치

```
sudo apt install nginx
```

3. nginx 설정
   - default.conf 파일 만들어서

```
server {
  listen 80;
  server_name j9b202.p.ssafy.io;
  return 301 https://j9b202.p.ssafy.io$request_uri;
}

server {
  listen 443 ssl;
  server_name j9b202.p.ssafy.io;

  ssl_certificate /etc/letsencrypt/live/j9b202.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/j9b202.p.ssafy.io/privkey.pem;
}
```

4. nginx 시작

```
systemctl start nginx
```

redis_password : k3s2ssafyb2020902!

[MySql 설정]
username: ssafyb202
password: k3s2b202ssafy

Docker Compose 설치

- 최신 버전을 가져오기 위한 jq 라이브러리 설치

```
최신 버전을 가져오기 위한 jq 라이브러리 설치
```

- docker-compose 최신 버전 설치

```
$ VERSION=$(curl --silent https://api.github.com/repos/docker/compose/releases/latest | jq .name -r)
$ DESTINATION=/usr/bin/docker-compose
```

```
$ sudo curl -L https://github.com/docker/compose/releases/download/${VERSION}/docker-compose-$(uname -s)-$(uname -m) -o $DESTINATION
$ sudo chmod 755 $DESTINATION

// 터미널 재접속 하기!

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

- $ docker-compose -v
  - 설치가 올바르게 이루어졌는지 확인하기 위해 Docker Compose의 버전을 출력

MySQL설치

1. Docker Maria DB 이미지 다운로드 받기

```
$ docker pull mysql
```

2. Docker에 Maria DB 컨테이너 만들고 실행하기

```
$ docker run --name mysql -d -p 3306:3306 -v /var/lib/mysql_main:/var/lib/mysql --restart=always -e MYSQL_ROOT_PASSWORD=root mariadb
```

3. Maria DB에 database를 추가하고 user 권한 설정

- Docker - Mariadb 컨테이너 접속하기

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

- **MySQL** - 루트 계정으로 데이터베이스 접속하기

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

비밀번호는 "root"

MySQL 사용자 추가하기

```
예시) create user 'user_name'@'XXX.XXX.XXX.XXX' identified by 'user_password';

create user 'ssafy601'@'%' identified by 'ssafy601';
```

MySQL - 사용자 권한 부여하기

```
예시) grant all privileges on db_name.* to 'user_name'@'XXX.XXX.XXX.XXX';
flush privileges;

grant all privileges on *.* to 'ssafy202'@'%';
flush privileges;
```

MySQL - 데이터 베이스 만들기

```
예시) create database [db_name];

create database ssafy202;
```

레디스 설치

- Redis 이미지 받기

```
docker pull redis:alpine
```

- 도커 네트워크 생성 [디폴트값]

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

- 도커 네트워크 상세정보 확인

```
docker inspect redis-network
```

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

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

- Docker 컨테이너 확인

```
docker ps -a
```

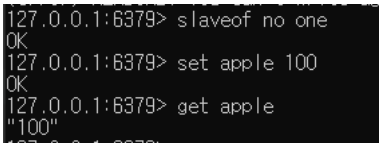
- 컨테이너 진입

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

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

- 권한 추가

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

- 테스트
  - OK 가 뜨면 성공



3. Dockerfile로 Jenkins images 받기 (Docker out of Docker, DooD 방식)

[Jenkins 컨테이너 안에서 직접적으로 Docker를 실행하지 않고 호스트 시스템의 Docker를 활용하여 컨테이너 관리 작업을 수행]

- Dockerfile 작성

```
# 폴더 생성
mkdir config && cd config

# 아래 내용 작
$ vi Dockerfile

FROM jenkins/jenkins:jdk17

#도커를 실행하기 위한 root 계정으로 전환
```

```
USER root

#Jenkins 컨테이너 내에서 Docker를 사용할 수 있도록 설정하는 부분
COPY docker_install.sh /docker_install.sh
RUN chmod +x /docker_install.sh
RUN /docker_install.sh

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

- docker, Jenkins 설정 shell 파일 (이또한 config 폴더에 vi docker_install.sh or vi docker.install.sh 파일을 만들어서 넣어야한다.

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

//GPG 키를 다운로드하고 저장소의 인증을 추가하는 것은 Docker의 소스가 신뢰할 수 있는지 확인
//시스템이 Docker 소프트웨어를 신뢰할 수 있도록 하는 과정
//즉, 해당 스크립트는 Docker 설치를 직접적으로 처리하지 않고, Docker 관련 설정을 수행하기 위한 작업
```

- Docker 이미지 생성

```
docker build -t jenkins/myjenkins . //현재 디렉토리에 있는 Dockerfile을 사용하여 젠킨스 이미지를 빌드하라는 의미
```

- Docker 볼륨 폴더 권한 설정

```
#디렉토리는 Jenkins 컨테이너에서 사용할 데이터 및 설정을 저장하기 위한 목적으로 사용
$ mkdir /var/jenkinsDir/
#Jenkins 컨테이너가 /var/jenkinsDir/ 디렉토리에 쓰기 및 읽기 권한을 가지게 됨
$ sudo chown 1000 /var/jenkinsDir/
```

- Jenkins 컨테이너 생성

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

jenkins/myjenkins
```

젠킨스 컨테이너 안에 docker-compose 설치하기!!

- 방식은 동일하다

- 옵션 설명

  `-d` : 는 백그라운드에서 실행을 의미

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

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

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

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

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

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

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

- 옵션 설명

  `-d` : 는 백그라운드에서 실행을 의미

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

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

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

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

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

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

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

Jenkins 초기 세팅 및 테스트 (호스트 시스템의 Docker 데몬과 컨테이너 내의 프로세스들이 통신하기위함!)

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

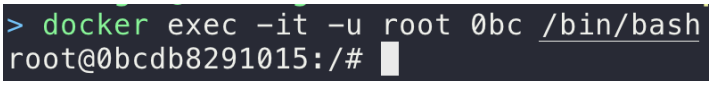
`?` 왜 이렇게하는가 ?

- `/var/run/docker.sock`은 Docker 데몬과 통신하기 위한 소켓 파일입니다. 이 소켓을 통해 호스트 시스템의 Docker 데몬과 컨테이너 내의 프로세스들이 통신합니다. 그런데 이 소켓 파일은 기본적으로 소유자와 그룹 모두 root 계정에게 속해 있습니다.
- Jenkins 컨테이너 내에서 Docker를 실행하기 위해서는 이 소켓 파일에 대한 적절한 권한 설정이 필요합니다. Jenkins 컨테이너 내의 Jenkins 프로세스가 Docker 데몬과 통신할 수 있도록 하려면 다음과 같은 이유로 해당 권한 설정을 수행합니다:
- 우리가 방금 생성한 컨테이너의 ID는 **0bcdb8~** 입니다. 도커는 다른 컨테이너 ID와 겹치지 않는 부분까지 입력하면 해당 컨테이너로 알아서 매핑해줍니다.

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

`exec`는 컨테이너에 명령어를 실행시키는 명령어인데, /bin/bash와 옵션 -it를 줌으로써 컨테이너의 쉘에 접속할 수 있습니다.

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



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

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

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

```
docker restart [컨테이너 ID]
```

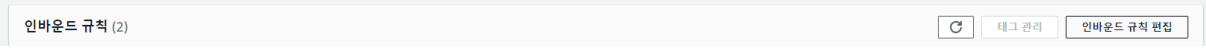
- Jenkins 패스워드 확인

```
docker logs [jenkins 컨테이너 ID]
```

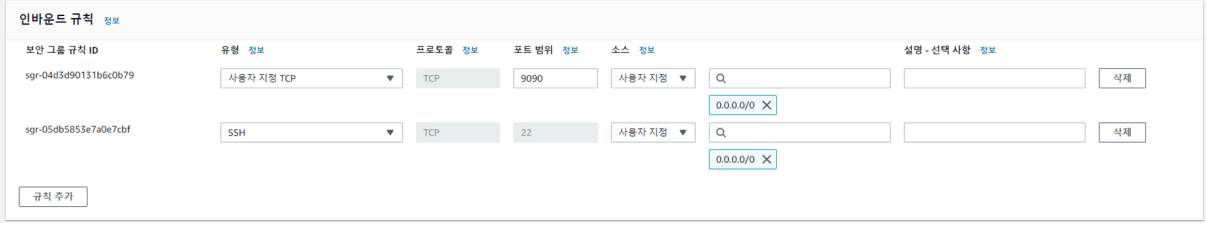
- docker logs 컨테이너 id를 입력해 로그를 출력하면 initialAdminPassword가 출력됩니다. 이 패스워드를 입력해주면 됩니다.

```
*************************************************************
*************************************************************
*************************************************************

Jenkins initial setup is required. An admin user has been created and a password generated.
Please use the following password to proceed to installation:

b99f67a2cf054174a73017e4498ce87d

This may also be found at: /var/jenkins_home/secrets/initialAdminPassword

*************************************************************
*************************************************************
*************************************************************
```

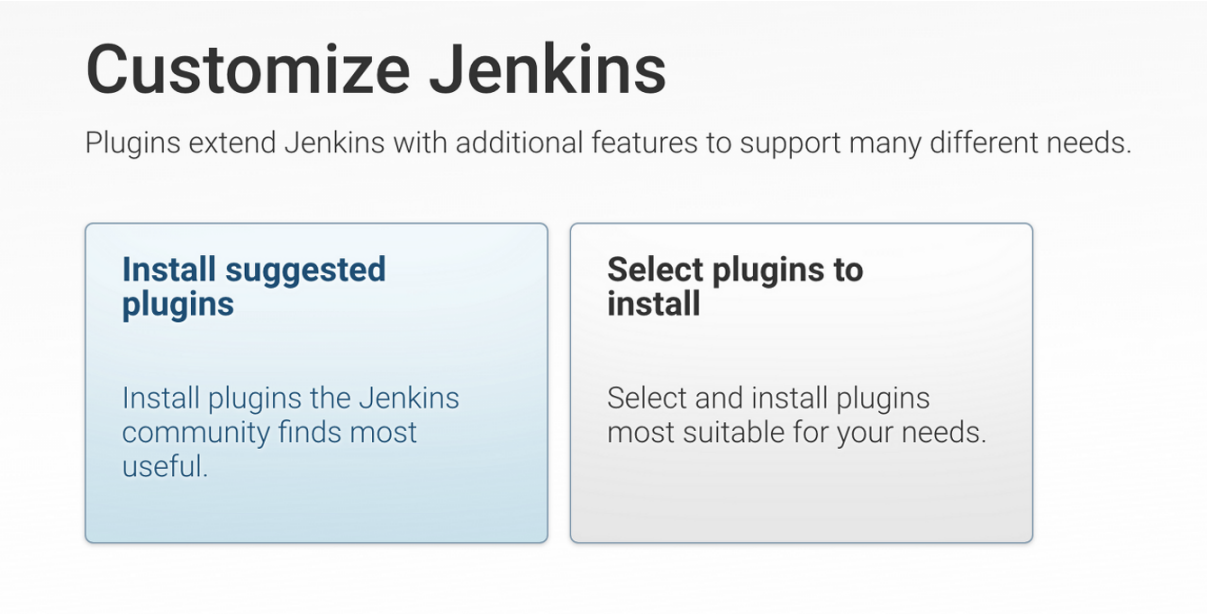
- 보안 그룹 설정을 해야 {public ip}:9090 에 접근할 수 있습니다.



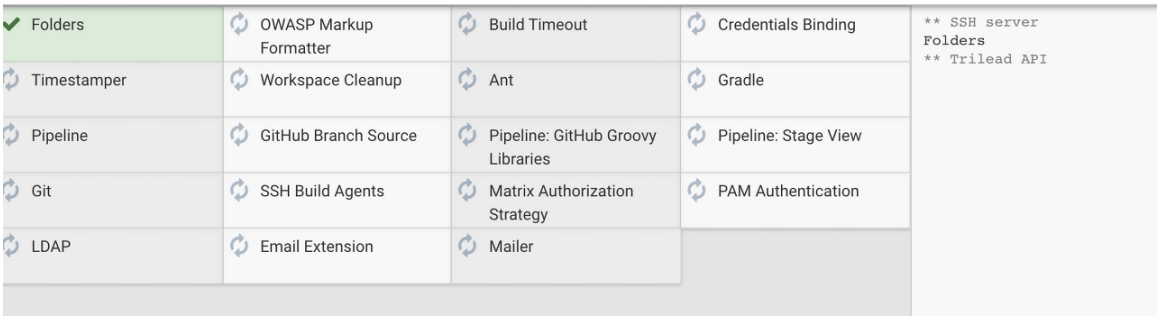
`인바운드 규칙 편집` 클릭



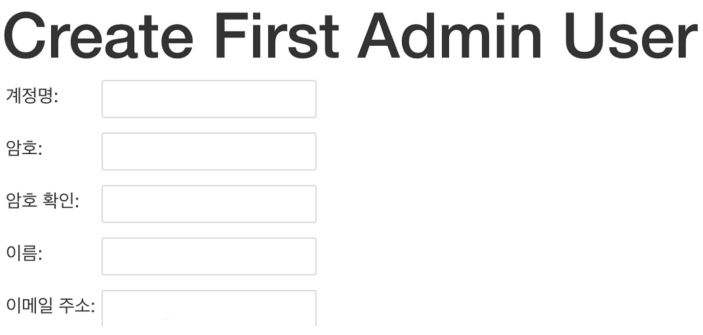
`규칙 추가` 클릭 → 유형: `TCP`, 포트 범위: {등록할 포트 번호}

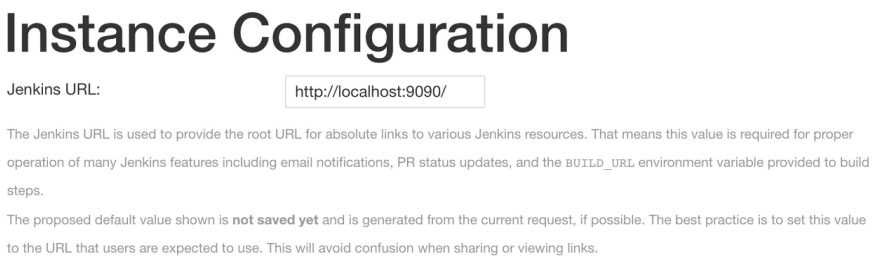


- 정상적으로 입력했다면 플러그인 설치가 나오는데, 우리는 Install suggested plugins를 선택합니다.



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





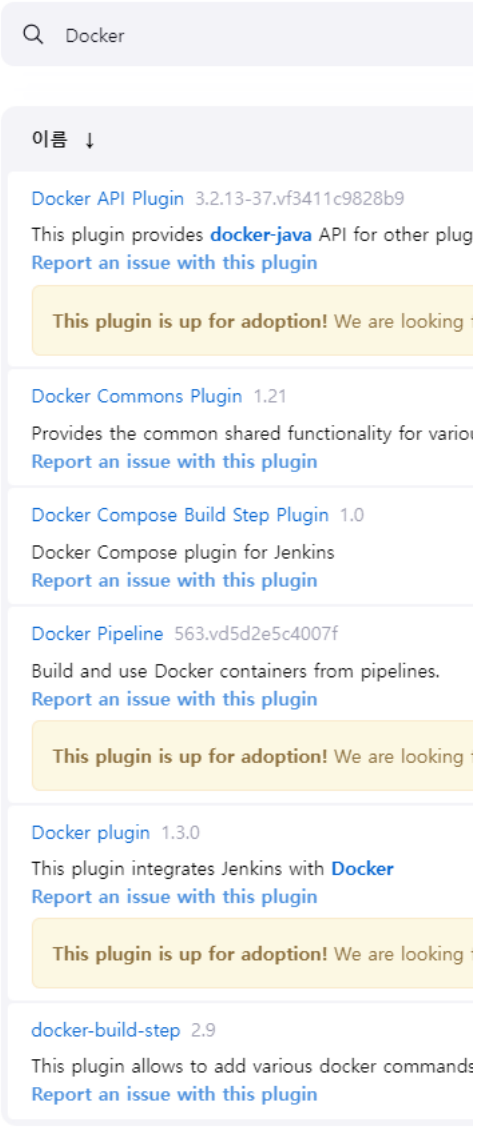
- 앞으로 이 url로 젠킨스에 접속하시면 됩니다.

Jenkins 플러그인 설정

- Gitlab, Docker 플러그인을 받습니다. (헷갈리다면 비슷한거라도 플러그인 다운받으세요, 너무 다른거 말고)

Gitlab

Docker



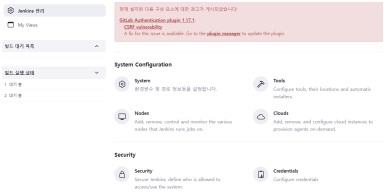
- 여기까지 오셨다면, 젠킨스 설치 및 초기 세팅 완료!

6. CI/CD (빌드 및 배포) 초기세팅 (먼저 Plugins에서 GitLab이랑 WebHook을다운받는다.)

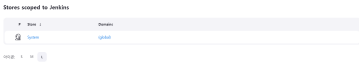
먼저 Jenkins 관리 → Credentials 를 누른다. → 먼저 만드는 이유는?

1. `Credentials를 먼저 만들어서 Jenkins 관리에서 관리하는 것은 보안을 강화하고 민감한 정보를 안전하게 저장하며, 효율적인 관리와 재사용성을 가능하게 하는 중요한 단계`
2. `즉 Credentials를 중앙화하여 관리함으로써 여러 프로젝트 또는 빌드에서 동일한 인증 정보를 사용할 수 있습니다. 변경이 필요한 경우, 한 곳에서 수정하면 모든 사용처에 즉시 적용됨.`

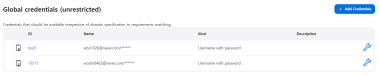
1. 먼저 Jenkins 관리 → Credentials 를 누른다.



2. Add Credentials를 누른다



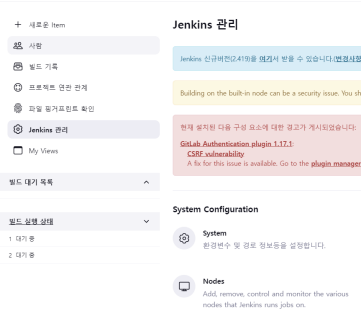
3. Add Credentials를 누른다

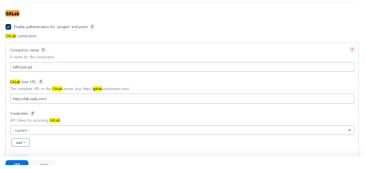


1. Kind는 Username with password 방식
2. UserName은 메일 주소 입력
3. lab.ssafy.com 계정 비밀번호를 입력한다.

2. GitLab Connection을 하기위해서 Jenkins 관리의 System을 누른다.

3. Connection은 자유
4. GitLab Host URL에서 lab.ssafy.com 입력

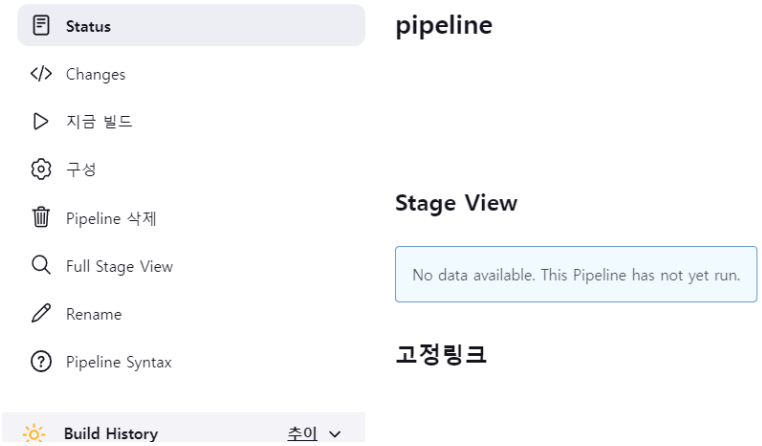


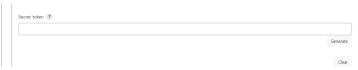


1. Pipeline에서 구성을 선택한다.

2. Build when a change is pushed to Gitlab~ 체크

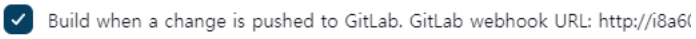
3. WebHook 등록하기위한 Secret Token 생성
4. 이걸로 WebHook으로 자동 감지하게 만들어야함.





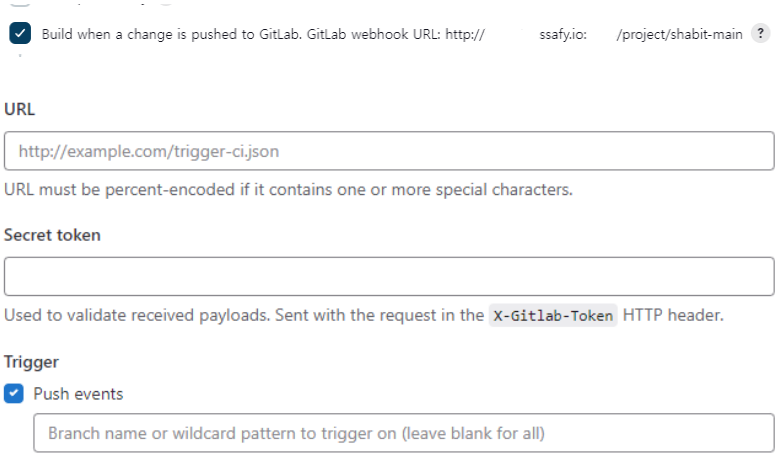
Webhook 설정

- Jenkins 트리거 체크

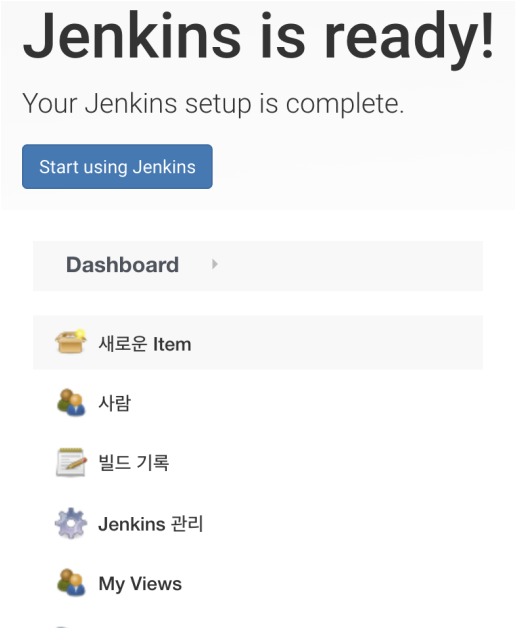


  - 트리거 체크 아래에 고급 누르면 됨 Jenkins 에서 `빌드 유발` → `Build when …` → `고급` → 하단에 `Secret token Generate` → 토큰 발급 완료!

- Gitlab Webhook에서 해당 토큰을 등록합니다.
  1. lab.ssafy.com Gitlab 프로젝트 접속
     a. 좌측 Settings → Webhook 접속
  2. Jenkins Project URL(jenkins 트리거 체크에서 써져 있는 webhook URL : 불라불라 이거임) 입력 후, Secret Token (jenkins 트리거 체크에서 발급 받은 토큰) 입력 그리고 Trigger는 원하는 Event에 대해서만 설정
  3. 생성후 Test를 눌렀을 때 200 응답이 return되면 성공



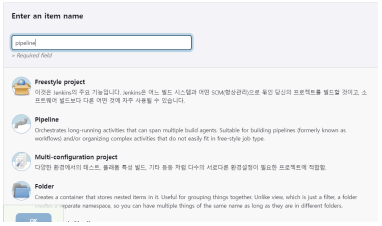
7. CI/CD (빌드 및 배포) 세팅

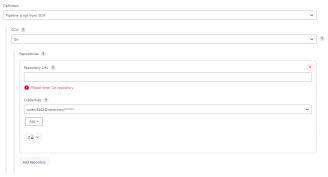


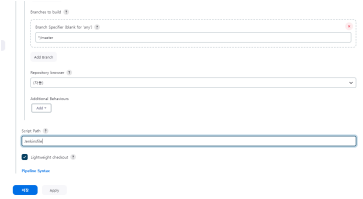
- 먼저, 대쉬보드의 파이프라인 을 클릭합니다.
- FreeStyle Project는 요즘은 많이 안쓴다고 들음… 파이프라인을 많이 쓰는이유는

요즘은 코드 기반의 자동화를 선호하며, 파이프라인이 더 널리 사용되고 있습니다.
파이프라인을 사용하면 코드와 인프라 구성을 함께 관리하며, 배포 파이프라인을 수정하거나 공유하기가 더 간편합니다. 라고합니다.

1. 파이프라인 선택
2. SCM Git 설정 , URL 설정, Credentials 설정
3. 병합할 브랜치 선택, Script Path는 JenkinsFile로 다룰예정







SSL 발급 받기

- ngnix가 설치되어있는 상태에서 시작하겠습니다 (nginx 설치는 다음장에)
- `sudo certbot certonly --standalone` 실행
  - 80번포트가 개방되어야함 -> 이미 실행중인 서비스, 데몬이 존재시, 이를 중단해야함
  - 자동 갱신 가능
  - 와일드카드 서브도매인 사용불가 (*.example.com)
  - 도메인이 자신의 서버에 연결되어야함 (A레코드를 자신의 서버로)
  - 와일드카드 서브도메인 인증서가 필요하지 않다면 권장

```
dev@vultr:~$ sudo certbot certonly --standalone
[sudo] password for dev:   <root password>
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator standalone, Installer None
```

```
Enter email address (used for urgent renewal and security notices)
 (Enter 'c' to cancel): vompressor@gmail.com

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf. You must
agree in order to register with the ACME server. Do you agree?
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: Y <- ACME 약관에 동의하는지 N선택시 진행불가

- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
Would you be willing, once your first certificate is successfully issued, to

share your email address with the Electronic Frontier Foundation, a founding
partner of the Let's Encrypt project and the non-profit organization that
develops Certbot? We'd like to send you email about our work encrypting the web,
EFF news, campaigns, and ways to support digital freedom.
- - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - - -
(Y)es/(N)o: N <- 이메일을 통해 Let's Encrypt 프로젝트 정보를 받아볼지
Please enter in your domain name(s) (comma and/or space separated)
(Enter 'c' to cancel): vompressor.com www.vompressor.com  <- {1} 인증서를 발급할 도메인 입력
Requesting a certificate for vompressor.com and www.vompressor.com

IMPORTANT NOTES:
 - Congratulations! Your certificate and chain have been saved at:
   /etc/letsencrypt/live/vompressor.com/fullchain.pem <- {2} 발급된 인증서 경로
   Your key file has been saved at:
   /etc/letsencrypt/live/vompressor.com/privkey.pem <- {2} 발급된 인증서 경로
   Your certificate will expire on 2021-05-16. To obtain a new or
   tweaked version of this certificate in the future, simply run
   certbot again. To non-interactively renew *all* of your
   certificates, run "certbot renew"
 - If you like Certbot, please consider supporting our work by:

   Donating to ISRG / Let's Encrypt:   https://letsencrypt.org/donate
   Donating to EFF:                    https://eff.org/donate-le
```

배포 시작

빌드 순서는 JenkinsFile→ start.prod.sh → docker-compose-prod.yml → Dockerfile

- 이 모두는 프로젝트내에 존재한다.
- 실행되는 순서는 BackEnd → FrontEnd → nginx 순으로 올라간다.

JenkinsFile (stages[큰 묶음] → stage[진짜 실행되는 작은 묶음])

```
pipeline {

    agent any
        stages {

            stage('Prepare') {
                steps {
                    sh 'echo "Clonning Repository"'
                    git branch: 'develop',
                        url: 'https://lab.ssafy.com/s09-bigdata-recom-sub2/S09P22B202.git',
                        credentialsId: 'f473f437-e90b-498e-a2fd-c6ccbf464b3a'
                }
                post {
                    success {
                        sh 'echo "Successfully Cloned Repository"'
                    }
                    failure {
                        sh 'echo "Fail Cloned Repository"'
                    }
                }
            }


            // stage('[BE]Bulid Gradle') {
            //     steps {
            //         sh 'echo "Bulid Gradle Start"'
            //         dir('BE') {

            //         }
```

```
            //     }
            //     post {
            //          failure {
            //              sh 'echo "Bulid Gradle Fail"'
            //          }
            //     }
            // }
            stage('Docker stop'){
                steps {
                    dir('BE'){
                        sh 'echo "Docker Container Stop"'


                        sh '''
                        result=$( docker container ls -a --filter "name=newsum*" -q )
                        if [ -n "$result" ]
                        then
                            docker stop $(docker container ls -a --filter "name=newsum*" -q)
                        else
                            echo "No stop containers"
                        fi
                        '''

                        //pwd
        //              도커 컴포즈 다운
                        //sh 'curl -L https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)
        //              해당 도커 컴포즈 다운한 경로로 권한 설정
                        //sh 'sudo chmod -R 777 /usr/local/bin'
                        //sh 'chmod +x /usr/local/bin/docker-compose'
                        //sh 'sudo mv /usr/local/bin/docker-compose /usr/local/bin/
        //              기존 백그라운드에 돌아가던 컨테이너 중지
                                        //기존 백그라운드에 돌아가던 컨테이너들을 DooD 방식으로 다운시킴.
                        sh 'docker-compose -f /var/jenkins_home/workspace/pipeline/docker-compose-prod.yml down'
                        //sh '/usr/bin/docker-compose -f docker-compose-prod.yml down'
                    }


                }
                post {
                    failure {
                        sh 'echo "Docker Fail"'
                    }
                }
            }

stage('RM Docker') {
    steps {
        sh 'echo "Remove Docker"'

        // 정지된 도커 컨테이너 찾아서 컨테이너 ID로 삭제함
        sh '''
            result=$( docker container ls -a --filter "name=newsum*" -q )
            if [ -n "$result" ]
            then
                docker rm $(docker container ls -a --filter "name=newsum*" -q)
            else
                echo "No such containers"
            fi
        '''

        // homesketcher로 시작하는 이미지 찾아서 삭제함
        sh '''
            result=$( docker images -f "reference=newsum*" -q )
            if [ -n "$result" ]
            then
                docker rmi -f $(docker images -f "reference=newsum*" -q)
            else
                echo "No such container images"
            fi
        '''

        // 안쓰는이미지 -> <none> 태그 이미지 찾아서 삭제함
        sh '''
            result=$(docker images -f "dangling=true" -q)
            if [ -n "$result" ]
            then
                docker rmi -f $(docker images -f "dangling=true" -q)
            else
                echo "No such container images"
            fi
        '''
    }
    post {
        failure {
```

```
            sh 'echo "Remove Fail"'
        }
    }
}



            stage('Set Permissions') {
                        steps {
                            // 스크립트 파일에 실행 권한 추가
                            sh 'chmod +x /var/jenkins_home/workspace/pipeline/start-prod.sh'
                        }
                   }
            stage('Execute start-prod.sh Script') {
                steps {
                    // start-prod.sh 스크립트 실행
                    sh '/var/jenkins_home/workspace/pipeline/start-prod.sh'
                }
            }



    //         stage('[FE] prepare') {
    //             steps {
    //                 dir('frontend'){
    //                     sh 'echo " Frontend Bulid Start"'
    //                     script {
    //                         sh 'docker-compose stop'
    //                         sh 'docker rm vue'
    //                         sh 'docker rmi frontend_vue'
    //                     }
    //                 }


    //             }

    //             post {
    //                 failure {
    //                     sh 'echo "Frontend Build Fail"'
    //                 }
    //             }
    //         }
    //         stage('Fronteend Build & Run') {
    //             steps {
    //                 dir('frontend'){
    //                     sh 'echo " Frontend Build and Start"'
    //                     script {

    // //                          업데이트된 코드로 빌드 및 실행
    //                         sh 'docker-compose up -d'
    //                     }
    //                 }


    //             }

    //             post {
    //                 failure {
    //                     sh 'echo "Bulid Docker Fail"'
    //                 }
    //             }
    //         }
        }
    }
```

start-prod.sh

```
#!/bin/bash

#pem_key_path="/c/Users/SSAFY/Desktop/I9B204T.pem"
#
## 접속할 EC2 인스턴스의 주소
#ec2_instance_address="i9b204.p.ssafy.io"



docker-compose -f docker-compose-prod.yml pull //현재 프로젝트에있는 docker-compose

COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose-prod.yml up --build -d
```

```
docker rmi -f $(docker images -f "dangling=true" -q) || true
```

- COMPOSE_DOCKER_CLI_BUILD=1 DOCKER_BUILDKIT=1 docker-compose -f docker-compose-prod.yml up --build -d
  - `docker-compose-prod.yml` 파일을 사용하여 정의된 서비스들을 빌드하고 컨테이너를 배포합니다.
  - `COMPOSE_DOCKER_CLI_BUILD`와 `DOCKER_BUILDKIT` 환경 변수를 설정하여 Docker Compose가 Docker CLI를 사용하여 빌드하고, 빌드 시 BuildKit을 사용하도록 지시합니다.
  - `-build` 옵션은 새 이미지를 빌드하도록 지시합니다.
  - `d` 옵션은 컨테이너를 백그라운드에서 실행하도록 합니다.
- `docker rmi -f $(docker images -f "dangling=true" -q) || true`:
  - 빌드 과정에서 생성되고 사용되지 않는 "dangling" 이미지를 강제로 삭제합니다.
  - `docker images -f "dangling=true" -q` 명령어를 통해 dangling 이미지들의 ID를 얻고, `f` 옵션은 필터를 적용하여 dangling 이미지만 선택합니다.
  - `q` 옵션은 이미지 ID만 표시하도록 합니다.
  - `|| true` 부분은 명령어 실행 중 에러가 발생해도 종료 코드가 0 (성공)으로 유지되도록 합니다. (에러가 없다면 실행 결과가 true이므로 무시됩니다.) `이 부분 조심 에러 안나도 빌드 됨!!!`

Docker-compose-prod.yml (여러 서비스 및 컨테이너를 정의하여 멀티 컨테이너 어플리케이션을 구축하고 실행하는 데 사용)

```
version: "3" #Compose 파일의 버전
services: # 서비스 정의를 시작합니다. 각 서비스는 별도의 컨테이너로 실행
  server:
    image: newsum-back:latest # 이미지 이름
    container_name: newsum_back # 컨테이너 이름
    build:
      context: ./BE/newsum # 컨테이너와 호스트 간의 포트 매핑을 설정합니다.

      args:
        SERVER_MODE: prod
    ports:
        - 8811:8811  #외부 Port : 내부 Port
    environment: #environment:: 컨테이너 내부의 환경 변수를 설정합니다.
        - TZ=Asia/Seoul
    networks:
      - newsum_network

#   server2:
#     image: newsum-back-chat:latest
#     container_name: ttp_back_chat
#     build:
#       context: ./BE/Talktopia/talktopia_chat
#       args:
#         SERVER_MODE: prod
#     ports:
#         - 15000:7500
#     environment:
#         - TZ=Asia/Seoul
#     networks:
#       - talktopia_network
  client:
    image: newsum-front:latest
    container_name: newsum_front
    build:
      context: ./FE/newsum
      dockerfile: Dockerfile #굳이안써도됨
    ports:
      - 3000:3000
    depends_on: # depends_on: 이 옵션은 서비스 간의 실행 순서와 의존성을 설정하는 데 사용, depends_on은 단순히 서비스의 실행 순서를 조정하고, 서비스가 실
      - server
    networks:
      - newsum_network #networks: 이 옵션은 서비스가 어떤 네트워크에 속하는지 정의하는 데 사용됩니다. 여러 서비스 간의 통신을 위해 사용되는 네트워크를 설정
#   nginx:
#     image: newsum-nginx:latest
#     container_name: newsum_nginx
#     build: ./.nginx
# #    depends_on:
```

```
# #      - server
#     ports:
#       - 80:80 # HTTP
#       - 443:443 # HTTPS
#     networks:
#       - newsum_network
#  redis:
#    image: redis
#    container_name: redis
#    hostname: talktopia.site
#    ports:
#      - 6379:6379
#    networks:
#      - talktopia_network
networks:
  newsum_network:
    driver: bridge
```

DockerFile - BackEnd

```
#FROM openjdk:11
#VOLUME /tmp
#EXPOSE 8000
#ARG JAR_FILE=build/libs/newsum-0.0.1-SNAPSHOT.jar
#COPY ${JAR_FILE} /app.jar
#ENTRYPOINT ["java","-jar","/app.jar"]
#ENV TZ=Asia/Seoul

FROM openjdk:11 as builder

COPY gradlew .
COPY gradle gradle
COPY build.gradle .
COPY settings.gradle .
COPY src src
RUN chmod +x ./gradlew
RUN ./gradlew bootJar

FROM openjdk:11
COPY --from=builder build/libs/*.jar app.jar
EXPOSE 8811

ARG SERVER_MODE
RUN echo "$SERVER_MODE"
ENV SERVER_MODE=$SERVER_MODE

ENTRYPOINT ["java", "-Dspring.profiles.active=${SERVER_MODE}","-Duser.timezone=Asia/Seoul", "-jar", "/app.jar"]
```

DockerFile - FrontEnd

```
FROM node:16.14.0-alpine as builder


# 작업 폴더를 만들고 npm 설치
WORKDIR /usr/src/app
COPY package.json /usr/src/app/package.json


RUN npm install --force

# 소스를 작업폴더로 복사하고 빌드
COPY . /usr/src/app
RUN npm run build

FROM nginx:alpine
# nginx의 기본 설정을 삭제하고 앱에서 설정한 파일을 복사
RUN rm /etc/nginx/conf.d/default.conf
COPY nginx/nginx.conf /etc/nginx/conf.d



# 위에서 생성한 앱의 빌드산출물을 nginx의 샘플 앱이 사용하던 폴더로 이동

COPY --from=builder /usr/src/app/build /usr/share/nginx/html
```

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

Nginx 설정

conf.d 폴더의 default.conf file

- `Nginx는 웹 서버 소프트웨어로서 클라이언트의 요청을 받아 정적 파일을 서비스하거나, 리버스 프록시 서버로 동작하여 백엔드 애플리케이션 서버로 요청을 전달하는 역할`

```
server {
    listen 80;
    server_name j9b202.p.ssafy.io;
    return 301 https://$host$request_uri;
}

server {
    listen 443 ssl;
    server_name j9b202.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j9b202.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j9b202.p.ssafy.io/privkey.pem;


    location / {
        proxy_pass http://localhost:3000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
    }


    location /api/ {
        proxy_pass http://localhost:8811/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Original-URI $request_uri;
    }

}
```

Nginx 파일 (Front-End) Nginx.conf

```
server {
  listen 3000;
  location / {
    root   /usr/share/nginx/html;
    index  index.html index.htm;
    try_files $uri $uri/ /index.html;
  }
  error_page   500 502 503 504  /50x.html;
  location = /50x.html {

      root   /usr/share/nginx/html;

  }
}
```