



포팅메뉴얼

배포 매뉴얼

EC2 기본

Docker 설치

- 사전 패키지 설치
- gpg 키 다운로드
- Docker 설치

DB

mysql 컨테이너 실행

Jenkins

젠킨스 설치(도커 컨테이너) 및 계정 생성

- docker-compose 이용 젠킨스 컨테이너 생성
- 젠킨스 계정 생성 및 플러그인 설치

젠킨스 프로젝트 생성 WebHook 설정, 자동 빌드 테스트

- 젠킨스 프로젝트 생성

GitLab Webhook

- 깃랩 WebHook 연결
- 연결 테스트

배포

젠킨스와 연결된 gitlab 프로젝트로 도커 이미지 빌드하기

- 젠킨스 bash shell 접근
- 사전 패키지 설치
- gpg 키 다운로드
- Docker 설치
- DockerFile 작성
- 젠킨스에서 DockerFile 이용 도커 이미지 생성

SSH 명령어 전송을 통해 빌드한 도커 이미지를 베이스로 컨테이너 생성

- 젠킨스 SSH 연결 설정 (Publish over SSH)
- 젠킨스 빌드 후 조치로 SSH 명령어 전송(EC2에 도커 컨테이너 생성)
- 빌드테스트

Nginx

Nginx를 통해 React와 Spring 경로 설정

- nginx.conf 파일 생성

Nginx Https 적용

- nginx 관련 명령어

배포 매뉴얼

Spring, React, Nginx를 이용하여 CICD 무중단 배포를 구축하는 방법

1. Gitlab Push Event가 일어나면
2. Jenkins에서 WebHook을 통해 자동으로 빌드를 실행
3. Jenkins에서 각각의 React(Nginx), Spring 프로젝트 내부의 DockerFile를 이용하여 Dockerimage 생성(tar 압축파일)
4. Jenkins에서 SSH 연결을 통해 AWS에 DockerContainer 생성
5. 외부에서 접속 : 도커 컨테이너에 올라간 Nginx에서 React와 Spring을 각각 '/', '/api'로 구분지어 연결

EC2 기본

Docker 설치

참고링크

[Docker] Ubuntu에 Docker 설치하기

커널 버전 확인: 리눅스 커널이 최소 3.10 버전 이상이어야한다. 아래와 같이 확인 가능하다. sudo 권한 혹은 root 권한을 소유한 계정에서 설치 진행 최신 배포판 OS인지 확인. 2020년 10월 기준 우분투 16.04, 18.04(LTS), 20.04(LTS) 지원 22년 2월 기준으로 확인결과, 2021년 4월 30일부터 우분투 16.04 LTS는 더 이상의 도커 릴리즈

 <https://dongle94.github.io/docker/docker-ubuntu-install/>



👉 사전 패키지 설치

```
sudo apt update
sudo apt-get install -y ca-certificates \
    curl \
    software-properties-common \
    apt-transport-https \
    gnupg \
    lsb-release
```

👉 gpg 키 다운로드

```
sudo mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
    "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
    $(lsb_release -cs) stable" | sudo tee /etc/apt/sources.list.d/docker.list > /dev/null
```

👉 Docker 설치

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

DB

```
sudo docker pull mysql
```

▼ MySQL env

- MYSQL_ROOT_PASSWORD=PassW@rd!! (Mysql 관리자 비밀번호, 관리자 계정은 root)
- MYSQL_USER=tomcat (Mysql 사용자 계정)
- MYSQL_PASSWORD=PassW@rd!! (Mysql 사용자 계정 비밀번호)

mysql 컨테이너 실행

```
sudo docker run --name {mysql컨테이너이름} -e MYSQL_ROOT_PASSWORD={루트 비밀번호} \
-e MYSQL_DATABASE={스키마이름} -e MYSQL_USER={USERNAME} \
-e MYSQL_PASSWORD={PASSWORD} -d -p3306:3306 mysql:latest
```

- workbench에서 연결 후 DB 관리 가능

Jenkins

젠킨스 설치(도커 컨테이너) 및 계정 생성

👉 docker-compose 이용 젠킨스 컨테이너 생성

```
vim docker-compose.yml
```

docker-compose.yml

```
version: '3'

services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock
      - /jenkins:/var/jenkins_home
    ports:
      - "9090:8080"
    privileged: true
    user: root
```

- services : 컨테이너 서비스
- jenkins : 서비스 이름
- image : 컨테이너 생성시 사용할 image, 여기서는 jenkins/jenkins:lts 이미지를 사용(jenkins의 lts버전을 가져온다는 뜻)
- container_name : 컨테이너 이름
- volumes : 공유 폴더 느낌, aws의 /var/run/docker.sock와 컨테이너 내부의 /var/run/docker.sock를 연결, /jenkins 폴더와 /var/jenkins_home 폴더를 연결.
- ports : 포트 매핑, aws의 9090 포트와 컨테이너의 8080 포트를 연결한다.
- privileged : 컨테이너 시스템의 주요 자원에 연결할 수 있게 하는 것 기본적으로 False로 한다고 한다.
- user : 젠킨스에 접속할 유저 계정 (root로 할 경우 관리자)

그리고 파일 작성을 완료하셨다면 ESC를 눌러주세요.

뭔가 파일이 잘못 작성되어

저장하지 않고 끝내는 :q!

저장하고 끝내는 :wq를 입력해주시면 됩니다.

컨테이너 생성

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

👉 젠킨스 계정 생성 및 플러그인 설치

1. Administrator Password 발급

```
sudo docker logs jenkins
```

2. 서버 공인 IP:9090 포트로 접속 후 입력
3. 두 개의 버튼 중 `Install suggested plugins` 클릭
4. 젠킨스 계정 생성 form 입력 후 `Save and Continue` 클릭
5. `Save and Finish`, `Start using Jenkins` 버튼으로 젠킨스 시작
6. jenkins 관리 탭 클릭 → 플러그인 관리 페이지로 이동
7. 설치 가능 탭으로 탭 변경 후 검색어에 `gitlab` 을 검색
 - a. GitLab / Generic Webhook Trigger / Gitlab API / GitLab Authentication 체크
 - b. `install without restart` 클릭
8. Docker 검색
 - a. Docker / Docker Commons / Docker Pipeline / Docker API 체크
 - b. `install without restart` 클릭
9. SSH 검색
 - a. Publish Over SSH 체크
 - b. `install without restart` 클릭

젠킨스 프로젝트 생성 WebHook 설정, 자동 빌드 테스트

참고링크

CI/CD #8. Gitlab Webhook으로 Jenkins 빌드 유발하기

webhook 기술을 통해 Gitlab과 Jenkins Build 유발하기 앞선 포스팅 을 통해 gitlab과 jenkins를 연동하고 gitlab저장소에 있는 소스를 기반으로 Maven 빌드하는 과정을 기술했었다. 이번 포스팅은 webhook기술에 대해 알아보고 해당 기술을 통해 gitlab에서 소스 변동이 발생될 시 Jenkins에 자동으로 빌드가 되게끔 구현해본다.

🔗 <https://zunoxi.tistory.com/106>



👉 젠킨스 프로젝트 생성

1. 젠킨스 메인페이지에서 `새로운 item` 클릭
2. 프로젝트 이름 작성, `Freestyle project` 를 클릭하고 `OK` 버튼을 클릭
3. `소스코드 관리` → git 라디오 버튼 클릭
 - a. `Repository URL` 에 Git 레포지토리 URL을 입력 (에러박박이 정상)
 - b. `Credentials` 에서, `add` → `jenkins` 클릭
 - Username : 싸피킷 아이디
 - Password : 싸피킷 비밀번호
 - ID : Credential 구별할 아무 텍스트
 - c. `Credentials` 에서 이제 만들어진 `Credential` 을 선택 → 오류메시지가 사라지면 성공

소스 코드 관리

☐ None

☒ Git ?

Repositories ?

Repository URL ?

https://lab.ssfy.com/s07-blockchain-nft-sub2/S07P22A607.git

Credentials ?

jilwon0297@naver.com/*****

+ Add

고급...

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/develop

Add Branch

Repository browser ?

(자동)

Additional Behaviours

Add ▾

4. 빌드 유발

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://j7a607.p.ssfy.io:9090/project/trippiece ?

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

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급...

☐ Generic Webhook Trigger ?

☐ GitHub hook trigger for GITScm polling ?

☐ Poll SCM ?

- a. 체크
 - b. 고급 → **Secret Token** Generate → 저장해두기
5. Build
- a. **Add build step** 클릭 → **Execute Shell** 선택
 - b. 연결만 테스트하는 것이기 때문에 일단 `pwd` 명령어를 입력 (추후 수정)
6. 저장
7. **지금 빌드** → 젠킨스 수동빌드 진행 후 성공 확인
8. 빌드 히스토리 → **Console Output** → 작동 확인

GitLab Webhook

👉 깃랩 WebHook 연결

- 깃랩 **Repository** → **WebHooks**
▼ **Settings** → **Webhooks**
- URL에 `http://배포서버공인IP:9090/project/생성한jenkins프로젝트이름/` 입력

☐ Build after other projects are built ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: `http://k7a307.p.ssafy.io:9090/project/frontend` ?

Enabled GitLab triggers

☒ Push Events

- Secret token → 젠킨스 프로젝트를 생성할 때 저장해둔 값 입력
- 빌드 유발 Trigger → **Push events** 설정
- 대상 Branch → **원하는 브랜치명** 으로 설정
- Add Webhook**

👉 연결 테스트

- 생성된 WebHook에서 **test** → **Push events**
- code 200** 확인

배포

젠킨스와 연결된 gitlab 프로젝트로 도커 이미지 빌드하기

배포 자동화 (2) 백엔드, 프론트엔드 도커 이미지 빌드

(1) Docker 설치 및 Jenkins 설정 (2) 백엔드, 프론트 엔드 도커 이미지 빌드 (3) 원격 서버에서 도커 이미지 실행: Jenkins 파이프라인 작성 (4) HTTPS 적용 및 Nginx 설정 : letsencrypt와 certbot을 사용한 SSL 인증서 설치, 리버스 프록시 적용 지난 포스팅에서 배포 자동화를 한다고 이야기 했고 자동화를 위해 도커와 젠킨스를 설치했다.

👉 <https://sinawi.tistory.com/371>



참고링크

👉 젠킨스 bash shell 접근

```
sudo docker exec -it jenkins bash
```

👉 사전 패키지 설치

```
apt update
apt-get install -y ca-certificates \
  curl \
  software-properties-common \
  apt-transport-https \
  gnupg \
  lsb-release
```

👉 gpg 키 다운로드

```
mkdir -p /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o /etc/apt/keyrings/docker.gpg

echo \
  "deb [arch=$(dpkg --print-architecture) signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian \
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null
```

👉 Docker 설치

```
apt update
apt install docker-ce docker-ce-cli containerd.io docker-compose
```

여기까지 진행하면 **Jenkins Container** 에도 **Docker** 설치 완료

👉 DockerFile 작성

- FROM : 베이스 이미지를 지정
- WORKDIR : 작업 디렉토리 설정
- COPY : 파일 복사 <Host 파일 경로> <Docker 이미지 파일 경로>
- RUN : 명령 실행
- CMD 컨테이너 실행 명령
- EXPOSE : 포트 익스포트

Spring

```
FROM openjdk:11-jre-slim
WORKDIR app
EXPOSE 8080
COPY ./build/libs/backend-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT exec java $JAVA_OPTS -jar ./app.jar
```

React

```
FROM node:16.15.0 as build-stage
WORKDIR /var/jenkins_home/workspace/{젠킨스프로젝트이름}/{react_repo이름}
```

```

COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /var/jenkins_home/workspace/{젠킨스프로젝트이름}/{react_repo이름}/dist /usr/share/nginx/html

# Nginx 설정 (Nginx 설정 전에는 주석처리해두어야 예러가 나지 않음)
COPY --from=build-stage /var/jenkins_home/workspace/{젠킨스프로젝트이름}/{react_repo이름}/deploy_conf/nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

👉 젠킨스에서 DockerFile 이용 도커 이미지 생성

1. 젠킨스 프로젝트 페이지 → [구성](#)

2. [Build 탭](#)

a. pwd로 적어두었던 명령어를 다음 명령어로 변경 후 저장

```

docker image prune -a --force
mkdir -p /var/jenkins_home/images_tar

#frontend docker image
cd /var/jenkins_home/workspace/{젠킨스프로젝트이름}/{react_repo이름}/
docker build . -t react
docker save react > /var/jenkins_home/images_tar/react.tar

#backend docker image
cd /var/jenkins_home/workspace/{젠킨스프로젝트이름}/{spring_repo이름}/
chmod +x gradlew
./gradlew bootJar
docker build . -t spring
docker save spring > /var/jenkins_home/images_tar/spring.tar

ls /var/jenkins_home/images_tar

```

▼

```

FROM openjdk:11-jre-slim
WORKDIR app
EXPOSE 8080
COPY ./build/libs/backend-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT exec java $JAVA_OPTS -jar ./app.jar

```

```

FROM node:16.15.0 as build-stage
WORKDIR /var/jenkins_home/workspace/frontend/frontend
COPY package*.json ./
RUN npm install
COPY . .
RUN npm run build
FROM nginx:stable-alpine as production-stage

COPY --from=build-stage /var/jenkins_home/workspace/frontend/frontend/dist /usr/share/nginx/html

# Nginx 설정 (Nginx 설정 전에는 주석처리해두어야 예러가 나지 않음)
COPY --from=build-stage /var/jenkins_home/workspace/frontend/frontend/deploy_conf/nginx.conf /etc/nginx/conf.d/default.conf

EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]

```

- docker image prune -a --force : 사용하지 않는 이미지 삭제
- mkdir -p /var/jenkins_home/images_tar : 도커 이미지 압축파일을 저장할 폴더 생성
- docker build . -t react : 도커 이미지 빌드 (React 프로젝트)
- docker save react > /var/jenkins_home/images_tar/react.tar : 도커 이미지를 react.tar로 압축하여 위에서 생성한 폴더에 저장
- ./gradlew bootJar : java spring gradle buildjar

- `docker build -t spring` : 도커 이미지 빌드(spring 프로젝트)
 - `docker save spring > /var/jenkins_home/images_tar/spring.tar` : 도커 이미지를 spring.tar로 압축하여 위에서 생성한 폴더에 저장
 - `ls /var/jenkins_home/images_tar` : 해당 폴더에 있는 파일 목록 출력(잘 압축되어 저장되었는지)
3. 젠킨스 빌드 후 서버에서 `cd /jenkins/images_tar` 로 이동해 `ls` 로 tar파일이 잘 들어있는지 확인

```
ubuntu@ip-172-26-9-166:/jenkins/images_tar$ ls
react.tar
ubuntu@ip-172-26-9-166:/jenkins/images_tar$ ls
react.tar  spring.tar
ubuntu@ip-172-26-9-166:/jenkins/images_tar$
```

baXterm by subscribing to the professional edition here: <https://mohaxterm.mohatek.net>

제대로 빌드되면 이렇게 나옴

SSH 명령어 전송을 통해 빌드한 도커 이미지를 베이스로 컨테이너 생성

참고링크

AWS EC2 계정 암호설정하기 / Jenkins SSH를 이용해서 배포해보기

Jenkins를 설치한 EC2 instance의 설정이 거의 다 끝나갑니다. Jenkins가 설치되어있는 instance이외에도 다른 EC2 Instance를 생성합니다. 이 instance는 실제로 application의 서버를 올려서 구동할 것 입니다. (배포 대상 서버) 단지 SSH를 이용해서 jenkins가 설치된 A instance로 부터 배포 대상인 B instance로 빌드된 아이템을 전송하기에

<https://velog.io/@dahunyo/AWS-EC2-%EA%B3%84%EC%A0%95-%EC%95%94%ED%98%B8%EC%84%A4%EC%A0%95%ED%95%98%EA%B8%B0-Jenkins-SSH%EB%A5%BC-%EC%9D%B4%EC%9A%A9%ED%95%B4%EC%84%9C-%EB%B0%B0%ED%8F%AC%ED%95%B4%EB%B3%B4%EA%B8%B0>

AWS EC2 이것저것

👉 젠킨스 SSH 연결 설정 (Publish over SSH)

1. 젠킨스 홈페이지 → `Jenkins 관리` → `시스템 설정`
2. `Public over SSH` 항목 → `SSH Servers` 추가
 - a. 항목 작성
 - Name : 그냥 이름
 - Hostname : EC2 IP
 - Username : EC2 접속 계정 이름 // ubuntu
 - b. 고급버튼
 - a. `Use password authentication, or use different key` 체크
 - b. form → `Key` : 키 페어 pem 파일 내 텍스트 내용 전체 복사 후 붙여넣기
 - c. `Test Configuration` 버튼을 눌렀을 때 Success가 나오면 성공

👉 젠킨스 빌드 후 조치로 SSH 명령어 전송(EC2에 도커 컨테이너 생성)

1. 젠킨스 프로젝트 페이지 → `구성`
2. `빌드 후 조치 탭`
 - a. 빌드 후 조치 추가 → `Send build artifacts over SSH` 선택
 - b. `Source files` : 아무거나 작성
 - c. `Exec command` : 아래 명령어를 복사 붙여넣기

```
sudo docker load < /jenkins/images_tar/spring.tar
sudo docker load < /jenkins/images_tar/react.tar

sudo docker rm react -f
sudo docker rm spring -f

sudo docker run -d -p 3000:80 --name react react
```

```
echo "Run frontend"
sudo docker run -d -p 8080:8080 --link {mysql컨테이너이름} --name spring spring
echo "Run backend"
```

d. 저장



```
sudo docker load < /jenkins/images_tar/spring.tar
sudo docker rm spring -f

sudo docker run -d -p 8080:8080 -e JAVA_OPTS=-Djasypt.encryptor.password=zazuabi --link ssapin --name spring spring
echo "Run backend"
```

```
sudo docker load < /jenkins/images_tar/react.tar

sudo docker rm react -f

sudo docker run -d -p 3000:80 --name react react
echo "Run frontend"
```

👉 빌드테스트

빌드 버튼을 눌러 빌드해주면 콘솔에서 결과를 확인할 수 있고, 서버의 80포트(HTTP)에는 React를 8080포트에는 Spring을 서비스함을 볼 수 있다.

처음 설명했던 작동방식 1~5중에서 4번 까지 완료

Nginx

Nginx를 통해 React와 Spring 경로 설정

이 과정을 해놓지 않으면 Https 설정을 할 때 높은 확률로 번거로운 작업이 추가로 생길 것이고, 만약 프론트는 Https에 성공했는데 백엔드가 Https 적용에 실패한다면 Https -> http의 크로스 도메인 오류 때문에 백엔드 API를 불러올 수 없는 치명적인 오류도 생기게 됨

따라서 하나의 도메인, 한 개의 Port에서 두 서비스를 구분 짓는 부분이 필요함

기존 리액트와 포트가 분리되어 8080 포트를 이용해야 접속 가능한 백엔드 서비스를 80 포트를 통해 접속할 수 있도록 변경시켜주는 작업

👉 nginx.conf 파일 생성

1. nginx.conf 파일 생성 및 편집기 이동

```
cd /jenkins/workspace/{젠킨스프로젝트}
sudo mkdir deploy_conf
cd deploy_conf
sudo vim nginx.conf
```

2. nginx.conf 파일 편집

```
upstream backend{
    ip_hash;
    server [도메인 주소]:8080;
}

server {
    listen      80;
    listen     [::]:80;
    server_name localhost;

    #access_log /var/log/nginx/host.access.log  main;

    location / {
```

```

    root    /usr/share/nginx/html;
    index   index.html index.htm;
    proxy_hide_header Access-Control-Allow-Origin;
    add_header 'Access-Control-Allow-Origin' '*';
}

#error_page 404          /404.html;

# redirect server error pages to the static page /50x.html
#
error_page 500 502 503 504 /50x.html;
location = /50x.html {
    root    /usr/share/nginx/html;
}

# proxy the PHP scripts to Apache listening on 127.0.0.1:80
#
#location ~ /\.php$ {
#    proxy_pass http://127.0.0.1;
#}

# pass the PHP scripts to FastCGI server listening on 127.0.0.1:9000
#
#location ~ /\.php$ {
#    root    html;
#    fastcgi_pass 127.0.0.1:9000;
#    fastcgi_index index.php;
#    fastcgi_param SCRIPT_FILENAME /scripts$fastcgi_script_name;
#    include fastcgi_params;
#}

# deny access to .htaccess files, if Apache's document root
# concurs with nginx's one
#
#location ~ /\.ht {
#    deny all;
#}
}

```

- upstream 을 통해서 backend를 로컬 ip:8080 주소와 연결
 - 해당 주소를 location /api 에 연결
 - 기존 리액트 프로젝트는 location / 에 연결
 - 결과적으로 공인 ip주소/api로 요청을 하게 되면 Nginx에서 스프링서버로 연결을 시켜줌
 - nginx와 스프링 서버사이의 통신은 로컬에서 이루어지기 때문에 공인 IP를 등록할 필요가 없음
3. `nginx.conf` 파일 작성을 마쳤다면 `esc, :wq` 를 통해 파일을 저장해줍니다.

Nginx Https 적용

1. EC2에 Certbot 설치

```

sudo add-apt-repository ppa:certbot/certbot
sudo apt-get upgrade
sudo apt-get install python3-certbot-nginx

```

2. SSL 인증서 발급

```

sudo certbot certonly --nginx -d [도메인 주소]

```

3. SSL 관련 conf 수정

```

cd /etc/nginx/sites-available
sudo vim default

```

conf 파일

```

server {
    listen 80;
    listen [::]:80;
    server_name  [도메인 주소];

    location / {
        rewrite ^(.*) https://[도메인 주소]:443$1 permanent;
    }
}

server{
    listen 443 ssl;
    listen [::]:443 ssl;
    server_name  [도메인 주소];
    # underscores_in_headers on;

    #ssl config
    ssl_certificate /etc/letsencrypt/live/[도메인 주소]/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/[도메인 주소]/privkey.pem;

    location / {
        proxy_pass http://[도메인주소]:3000;
        proxy_hide_header Access-Control-Allow-Origin;
        add_header 'Access-Control-Allow-Origin' '*';
        proxy_pass_request_headers on;
    }

    location /api/ {
        proxy_pass http://[도메인주소]:8080/;
        proxy_hide_header Access-Control-Allow-Origin;
        add_header 'Access-Control-Allow-Origin' '*';
    }
}

```

👉 nginx 관련 명령어

상태확인

```
systemctl status nginx
```

```
systemctl status nginx
```

웹서버 정지

```
sudo systemctl stop nginx
```

웹서버 시작

```
sudo systemctl start nginx
```

웹서버 재시작

```
sudo systemctl restart nginx
```

설정 리로드

```
sudo systemctl reload nginx
```