



# 기술 가이드

2024.03.19

팀 명	TPlusT
이름	김건호

# 목차

목차.....	2
<b>I. 기술 가이드 활용법 .....</b>	<b>3</b>
기술 가이드 취지 .....	3
기술에 따른 기대효과 .....	3
활용시 유의사항 .....	4
기술 구성도 .....	5
흐름도 .....	5
<b>II. 기술 소개 .....</b>	<b>6</b>
Nginx.....	6
DNS .....	6
도메인 이름 시스템(DNS, Domain Name System).....	6
사용자가 웹 브라우저에서 도메인 이름(예: www.example.com)을 입력할 때, DNS 는 해당 이름을 인터넷 통신에 필요한 숫자로 이루어진 IP 주소로 변환한다. 이 과정을 통해 사용자는 복잡한 IP 주소를 기억하지 않고도 웹사이트에 쉽게 접근할 수 있게 한다.	
DNS 기능 .....	6
DNS 의 장점 .....	6
마이크로 아키텍처.....	7
마이크로 아키텍처의 장점 .....	7
마이크로 아키텍처의 구성요소 .....	8
Vagrant .....	8
Docker.....	8
Web/Was/DB.....	9
forward proxy .....	9
CI/CD .....	9
jenkins.....	10
gitlab .....	10
webhook.....	11
NFS.....	12
<b>III. 기술 구현을 위한 기본 설정 및 설치 .....</b>	<b>13</b>
Vagrant 를 이용한 가상환경 설치 .....	13
GNS 를 이용한 네트워크 구성 .....	15

<b>IV. 기술 구현 방법 .....</b>	<b>17</b>
Jenkins 서버 구축.....	30
gitlab 서버 구축 .....	31
Jenkins 의 인터넷 연결 문제로 인해 plugin 설치 불가.....	32
Jenkins-gitlab 연결 .....	36
gitlab 구성 .....	41
자동 빌드를 위한 webhook 설정 .....	44
Url is blocked: Requests to the local network are not allowed .....	46
파이프라인 자동화 코드 .....	47
NFS 서버 구축.....	55
<b>V. 출처 .....</b>	<b>57</b>

# I. 기술 가이드 활용법

---

## ● 기술 가이드 취지

모놀로식 아키텍처에서 마이크로 아키텍처로의 인프라 확충과 함께 기본적인 기술에서부터 최신 기술까지 많은 기술이 발전되었습니다.

이러한 방대한 기술을 정리하려는 노력은 난해한 개념 위주 설명과 이론적 절차 남발, 기술을 활용하려는 사용자의 가독성 침해 등 현재 발생하는 상황을 해결하기에는 한계가 있었습니다.

이에, 해당 문제로 인해 피해를 보고 있는 클라이언트의 입장에서는 한 눈에 대응 기술을 비교하여 장단점을 파악하고 도입 및 활용할 수 있는 실용적인 가이드가 절실했습니다.

본 기술 가이드는 기업들과 개인이 쉽게 이해하고 활용 가능하며, 또한 추가적인 대응 기술을 바로 도입할 수 있도록 하는 실용서가 되도록 아래와 같은 방향으로 작성되었습니다.

1. 난해한 기술적 분류가 아닌 활용 가능성 위주의 분류
2. 서비스 제공을 통한 현실적 문제 및 해결법
3. 현재 발생하는 문제점 개선을 위한 기술 위주로 설명
4. 기술적 배경의 생략
5. 기술 도입 시 장단점 소개

## ● 기술에 따른 기대효과

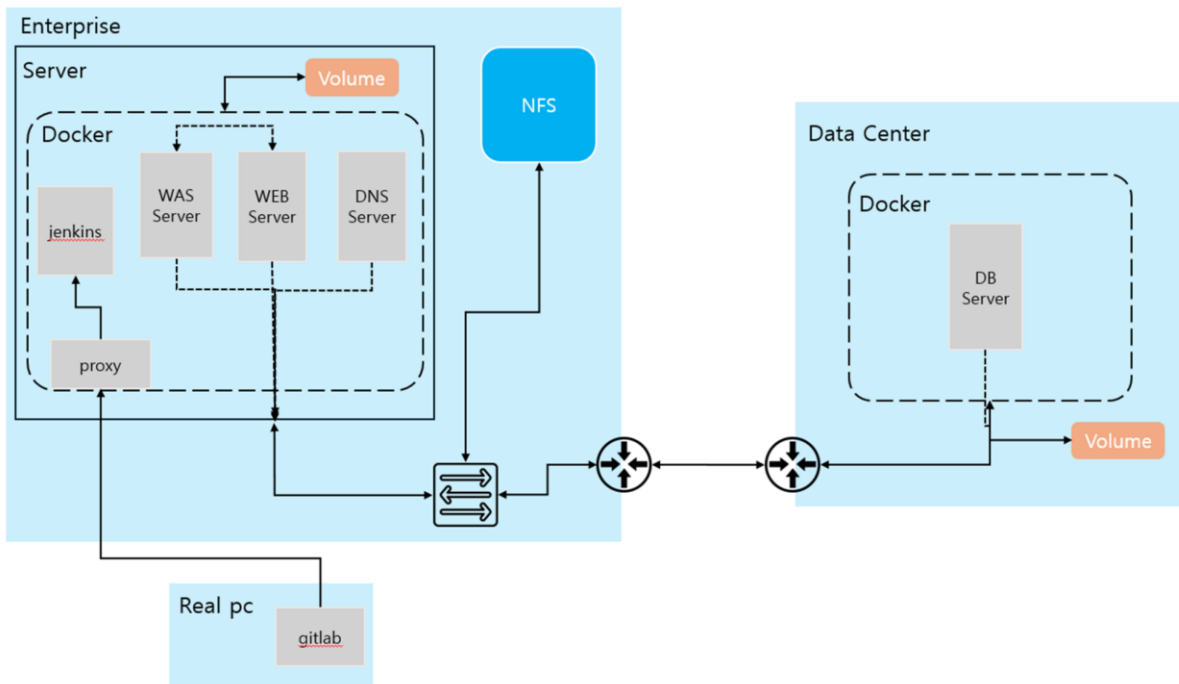
- 신서비스 확장성
- 개발과 운영의 협업 강화
- 빠른 시장 테스트

- 변화에 빠르게 대응
- 활용시 유의사항

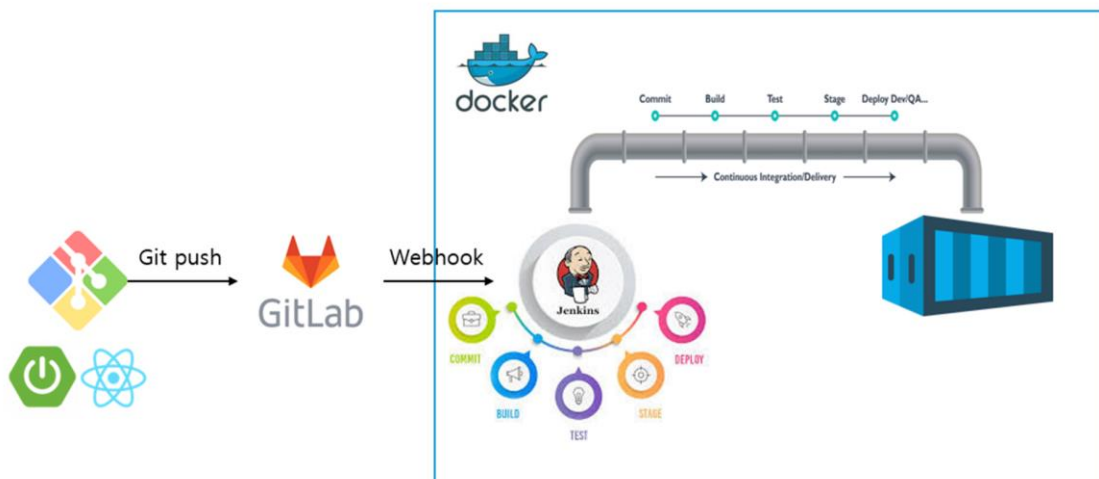
본 가이드를 활용하시는 사용자는 아래 사항을 유념하시기 바랍니다.

- 각 기업 혹은 사용자가 처한 상황을 고려할 필요
  - 대응 기술은 적용 되는 구조에 따라 나름 최선의 솔루션이 될 수 있으며 개별 기술을 평가하는 것은 한계가 있습니다.
  - 본 가이드는 대응 기술 도입하는 데 있어서 더 나은 이해와 최소한의 판단자료를 제공하기 위함입니다.
  - 이에 사용자는 본 자료가 일반론적인 내용이며, 발생하는 문제와 구조에 따라 달라짐을 인식하고, 해당 사용자가 처한 상황에 맞게 활용해야 합니다.
- 기술 간 융합 검토 필요
  - 개별 기술의 한계 극복을 위해 다른 기술과 같이 사용되는 경우가 많습니다.
  - 같이 사용되는 기술에 따른 경우의 수가 많아 모두 나열하여 소개하는 것이 불가능합니다.
  - 그러므로 본 자료는 각 기술의 기본적인 개념을 소개한 후 같이 사용된 사례를 설명할 수 있도록 작성되었습니다.
  - 반드시 소개된 사례가 아니더라도, 개별 기술 중 원하는 기술을 다른 기술과 사용하는 것은 가능하니 그 점을 이해하고 보시기를 추천드립니다.

## ● 기술 구성도



## ● 흐름도



## II. 기술 소개

---

- Nginx

Nginx 는 고성능의 웹 서버, 리버스 프록시, 그리고 이메일 프록시(POP3/IMAP) 기능을 제공하는 소프트웨어이다. 비동기 이벤트 기반의 구조로 설계되어, 매우 높은 동시 연결 처리능력을 가지고 있다. 정적 콘텐츠를 처리하는 데 있어 매우 빠른 속도를 보인다.

- DNS

도메인 이름 시스템(DNS, Domain Name System)

인터넷상의 자원을 찾기 위해 도메인 이름을 IP 주소로 변환하는 분산 데이터베이스 시스템이다.

사용자가 웹 브라우저에서 도메인 이름(예: `www.example.com`)을 입력할 때, **DNS** 는 해당 이름을 인터넷 통신에 필요한 숫자로 이루어진 **IP** 주소로 변환한다. 이 과정을 통해 사용자는 복잡한 IP 주소를 기억하지 않고도 웹사이트에 쉽게 접근할 수 있게 한다.

### DNS 기능

**DNS** 는 인터넷의 전화번호부와 같은 역할을 하며, 도메인 이름과 IP 주소 간의 매핑 정보를 관리한다. 이 시스템은 인터넷의 핵심 기능 중 하나로, 사용자가 웹사이트에 접근하거나 이메일을 보낼 때 필수적으로 사용된다.

### DNS 의 장점

- 사용 용이성

사용자가 웹사이트에 접근하기 위해 기억하기 쉬운 도메인 이름을 사용할 수 있게 한다.

- 분산 관리

DNS 는 전 세계에 분산된 서버 네트워크에 의해 관리된다. 이는 시스템의 탄력성을 높이고, 단일 지점에서의 장애가 전체 시스템에 영향을 미치는 것을 방지한다.

- 동적 업데이트

IP 주소가 변경되어도 도메인 이름은 동일하게 유지된다. DNS 서버는 변경된 IP 주소로 도메인 이름을 새로 매핑하여 사용자가 항상 올바른 사이트에 접근할 수 있도록 한다.

- 마이크로 아키텍처

현대의 웹 및 엔터프라이즈 애플리케이션 설계에서 널리 사용되는 설계 구조이다. 이 아키텍처는 애플리케이션을 세 개의 주요 계층으로 분리하여 개발과 관리의 복잡성을 줄이고, 시스템의 확장성, 유지보수성, 그리고 재사용성을 향상시킨다.

애플리케이션을 프리젠테이션 계층(Presentation Layer), 비즈니스 로직 계층(Business Logic Layer), 그리고 데이터 액세스 계층(Data Access Layer)으로 구분하는 설계 방식이다. 이러한 분리는 각 계층의 독립적인 개발과 수정을 가능하게 하며, 전체 시스템의 안정성과 확장성을 보장한다.

### 마이크로 아키텍처의 장점

- 유지보수성(Maintainability)

각 계층은 독립적으로 개발되고 유지보수될 수 있어, 변경 사항이나 업데이트가 필요할 때 전체 시스템을 재구성하지 않아도 된다.



- **확장성(Scalability)**  
비즈니스 로직이나 데이터 액세스 계층을 필요에 따라 확장하여 시스템의 처리 능력을 증가시킬 수 있다.
- **재사용성(Reusability)**  
표준화된 인터페이스를 통해 계층 간의 상호 작용이 이루어지므로, 다른 애플리케이션에서도 동일한 계층이나 컴포넌트를 재사용할 수 있다.
- **보안성(Security)**  
각 계층은 별도의 보안 메커니즘을 적용할 수 있으며, 특히 중요한 데이터를 다루는 데이터 액세스 계층에서 강화된 보안 조치를 취할 수 있다.

#### 마이크로 아키텍처의 구성요소

- **프리젠테이션 계층**  
사용자 인터페이스(UI)와 사용자 경험(UX)을 관리. 시스템과 사용자 간의 상호작용을 담당하며, 웹 브라우저나 모바일 앱을 통해 접근된다. 주로 모든 사용자에게 동일한 화면을 보여주는 기능을 한다.
- **비즈니스 로직 계층**  
애플리케이션의 핵심 기능과 비즈니스 규칙을 구현. 데이터의 처리와 계산, 비즈니스 규칙의 실행 등을 담당하며, 사용자마다 다른 화면을 보여주는 기능을 한다.
- **데이터 액세스 계층**  
데이터베이스나 다른 저장 시스템에 대한 접근을 제공. 데이터의 조회, 저장, 수정 등의 기능을 담당한다.

#### ● Vagrant

Vagrant 는 가상화 환경을 쉽게 생성하고 관리할 수 있는 도구이다. 개발 환경을 코드로 정의하여 버전 관리할 수 있으며, 일관된 개발 환경을 구축할 수 있다.

VirtualBox, VMware 와 같은 가상화 소프트웨어 위에서 작동하며, 이를 통해 개발자는 어떠한 환경에서도 동일한 작업 환경을 손쉽게 재현할 수 있다.

- Docker

Docker 는 애플리케이션을 컨테이너라는 격리된 환경 안에 패키징하여, 어떤 환경에서도 동일하게 실행할 수 있게 하는 오픈 소스 소프트웨어 플랫폼이다.

컨테이너는 가볍고 빠르며, OS 수준의 가상화를 제공하여

애플리케이션과 그 종속성을 함께 묶어 배포한다.

이를 통해 개발, 테스트, 배포 과정을 더욱 빠르고 효율적으로 만든다.

- Web/Was/DB

- **Web Server:** HTTP 프로토콜을 통해 웹 페이지나 파일 등의 정적 콘텐츠를 클라이언트(웹 브라우저)에게 전달하는 서버이다. Nginx, Apache 와 같은 소프트웨어가 이에 해당한다.
- **WAS (Web Application Server):** 웹 서버와 데이터베이스 서버 사이에서 사용자의 요청에 따라 애플리케이션을 실행하고, 그 결과를 웹 서버에 전달하는 역할을 한다. 동적 콘텐츠를 처리하기 위해 사용되며, Tomcat, JBoss 등이 여기에 속한다.
- **Database (DB):** 구조화된 데이터를 저장, 관리, 검색하기 위한 시스템이다. 웹 애플리케이션의 데이터를 저장하고, WAS 를 통해 처리된 요청에 따라 데이터를 검색하거나 수정한다. MySQL, PostgreSQL, MongoDB 등이 데이터베이스 관리 시스템(DBMS)의 예이다.

- **forward proxy**

포워드 프록시는 클라이언트와 인터넷 사이에 위치하는 서버이다.

클라이언트의 요청을 인터넷에 대신 전달하며, 결과를 클라이언트에게 반환한다.

이는 클라이언트의 실제 IP 주소를 숨기고 캐싱, 접근 제어, 감사 로깅 등의 기능을 제공한다.

포워드 프록시는 보안을 강화하고, 인터넷 사용을 모니터링 및 제어하는데 도움이 된다.

- **CI/CD**

지속적 통합(Continuous Integration, CI) 및 지속적 배포(Continuous Deployment, CD)는 소프트웨어 개발 프로세스를 자동화하여, 소프트웨어 품질을 개선하고 배포 시간을 단축하는 방법이다.

CI는 개발자가 작업한 코드를 중앙 저장소에 자주 병합하게 해서, 소프트웨어 통합 과정에서 나타날 수 있는 문제를 빨리 찾아내고 고치는 걸 목표로 한다. 이 과정에서 자동화된 빌드와 테스트가 진행되어 코드 품질을 지속적으로 체크한다.

CD는 CI의 다음 단계로, 자동화된 테스트를 통과한 코드를 프로덕션 환경으로 바로 배포하는 과정이다. 이 과정은 개발에서 배포까지의 전체 파이프라인을 자동화해, 소프트웨어를 더 자주, 더 신속하게 배포할 수 있게 해준다. CD를 통해 개발팀은 사용자 피드백을 빨리 받아들이고, 시장 변화에 신속하게 대응할 수 있다.

- **jenkins**

Jenkins는 오픈 소스 자동화 서버로, 주로 지속적 통합(CI) 및 지속적 배포(CD) 프로세스를 자동화하는 데 사용된다. 이 도구는 소프트웨어 개발의 빌드, 테스트, 배포 과정을 자동화하여 개발자들이 더 높은 품질의 소프트웨어를 더 빠르게 배포할 수 있도록 돕는다.

**Jenkins**의 핵심 장점 중 하나는 확장성이 뛰어난 플러그인 시스템을 제공한다는 것이다. 수천 가지의 플러그인을 통해 거의 모든 외부 시스템과 서비스를 **Jenkins**와 통합할 수 있어, 맞춤형 **CI/CD** 파이프라인을 구축하는 데 있어서의 유연성을 대폭 향상시킨다.

**Jenkins**를 사용함으로써 개발 팀은 코드 통합, 테스트 자동화, 빌드 및 배포 과정을 보다 체계적으로 관리할 수 있다. 이는 소프트웨어 개발 사이클을 단축시키고, 지속적인 피드백을 통해 문제를 신속하게 해결할 수 있게 만든다.

또한, **Jenkins**는 파이프라인 **as code** 기능을 지원하여, 파이프라인의 구성을 코드 형태로 관리할 수 있게 해준다. 이를 통해 파이프라인의 변경 사항을 버전 관리 시스템에서 추적하고, 변경 내용을 재사용하며, 공유할 수 있다.

- **gitlab**

**GitLab**은 웹 기반의 **Git** 저장소 관리 시스템으로, 코드 버전 관리 기능을 넘어서 프로젝트 관리의 다양한 측면을 지원한다. 이슈 추적, 코드 리뷰, **CI/CD** 파이프라인 구성 및 실행, 위키 등의 기능을 제공함으로써, 개발 프로세스의 협업과 자동화를 획기적으로 개선한다.

**GitLab**의 **CI/CD** 기능은 소프트웨어 개발의 자동화와 품질 관리를 강화하는 데 중요한 역할을 하며, 개발자가 코드를 커밋하는 즉시 자동으로 빌드, 테스트, 배포하는 것이 가능해진다.

**GitLab**은 소스 코드 관리뿐만 아니라 전체 **DevOps** 사이클을 커버할 수 있도록 설계되었다. 이는 개발, 테스트, 배포까지의 과정을 하나의 플랫폼에서 관리할 수 있게 함으로써, 팀 간의 협업을 강화하고 프로젝트 관리를 보다 효율적으로 만든다. 또한, **GitLab**은 클라우드 기반 서비스와 자체 서버에 설치해서 사용할 수 있는 옵션을 제공하여, 다양한 조직의 보안 요구사항과 운영 환경을 충족시킨다.

- **webhook**

**Webhook**은 특정 이벤트가 발생했을 때 사전에 설정된 **URL**로 **HTTP POST** 요청을 자동으로 보내는 방식이다. 이를 통해 개발자는 실시간으로 정보를 전달하거나, 서로 다른 시스템 간의 통합을 구현할 수 있다.

예를 들어, 코드 저장소에 새로운 커밋이 푸시되거나, 특정 조건에 따라 이슈가 생성될 때 자동으로 다른 시스템에 알림을 보내는 것이 가능하다. **Webhook**은 다양한 애플리케이션과 서비스에서 광범위하게 사용되며, 이는 시스템 간의 상호 작용을 자동화하고, 복잡한 워크플로를 구현하는데 있어 핵심적인 역할을 한다.

**Webhook**의 주요 장점 중 하나는 이벤트 기반의 통신을 가능하게 하여 시스템이 실시간으로 반응할 수 있게 한다는 점이다. 이는 예를 들어, 사용자가 폼을 제출하거나, 결제가 처리될 때 같은 즉각적인 반응이 필요한 경우에 매우 유용하다. 또한, **Webhook**은 간단하게 설정할 수 있으며, 복잡한 **API** 통합 없이도 빠르게 외부 시스템과의 연동을 구현할 수 있게 해 준다. 이러한 유연성과 효율성은 **Webhook**을 현대적인 애플리케이션 아키텍처에서 중요한 컴포넌트로 만든다.

- **NFS**

파일을 네트워크를 통해 여러 컴퓨터 간에 공유할 수 있게 해주는 프로토콜이다.

**NFS**를 사용하면 사용자는 네트워크 상의 다른 컴퓨터에 저장된 파일을 마치 로컬 파일 시스템에 있는 것처럼 접근하고 사용할 수 있다. 이는 파일 공유와 데이터 중앙 집중화에 유용하며, 특히 대규모 네트워크 환경에서 효과적이다.

### III. 기술 구현을 위한 기본 설정 및 설치

- Vagrant 를 이용한 가상환경 설치

#### Vagrantfile (가상머신 + Docker)

```
Vagrant.configure("2") do |config|
  # 첫 번째 가상 머신 설정 (combined_server)
  config.vm.define "combined_server" do |combined_config|
    combined_config.vm.box = "ubuntu/bionic64"
    combined_config.vm.network "private_network", ip: "192.168.30.10"
    combined_config.vm.provider "virtualbox" do |vb|
      vb.memory = "4096"
      vb.cpus = 2
    end

    combined_config.vm.provision "shell", inline: <<-SHELL
      # Docker 설치
      apt-get update
      apt-get install -y docker.io
      docker pull balabit/syslog-ng:3.36

      # WEB 서버 컨테이너 네트워크
      docker network create --subnet=172.18.0.0/24 --gateway=172.18.0.254 web
      # 내부 WAS 서버 컨테이너 네트워크 생성
      docker network create --internal --subnet=172.19.0.0/24 --gateway=172.19.0.254 was

      ## DNS 서버용 컨테이너 실행
      #docker run -d -it --name dns -p 53:53/udp ubuntu

    SHELL
  end

  #-----
  # 두 번째 가상 머신 설정 (db_server)
  config.vm.define "db_server" do |db_config|
    db_config.vm.box = "ubuntu/bionic64"
    db_config.vm.network "private_network", ip: "192.168.40.10"
    db_config.vm.provider "virtualbox" do |vb|
      vb.memory = "4096"
      vb.cpus = 2
    end

    db_config.vm.provision "shell", inline: <<-SHELL
      # Docker 설치 및 컨테이너 실행
      apt-get update
      apt-get install -y docker.io

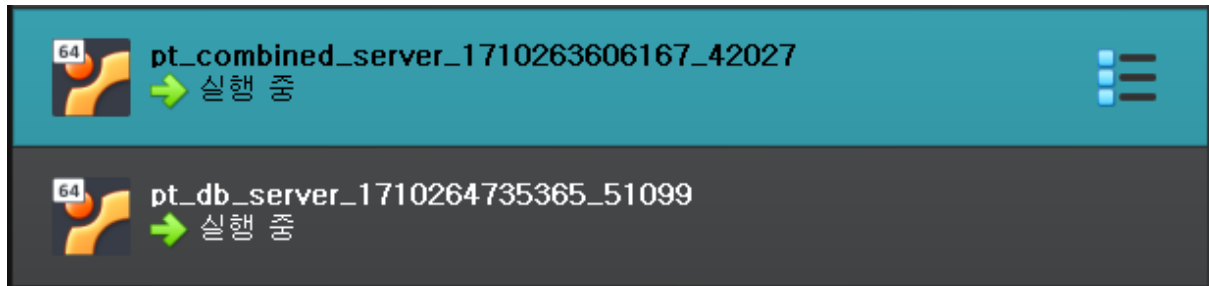
      # 데이터베이스 서버 컨테이너 네트워크
      docker network create --subnet=172.20.0.0/24 --gateway=172.20.0.254 db

    SHELL
  end
end
```

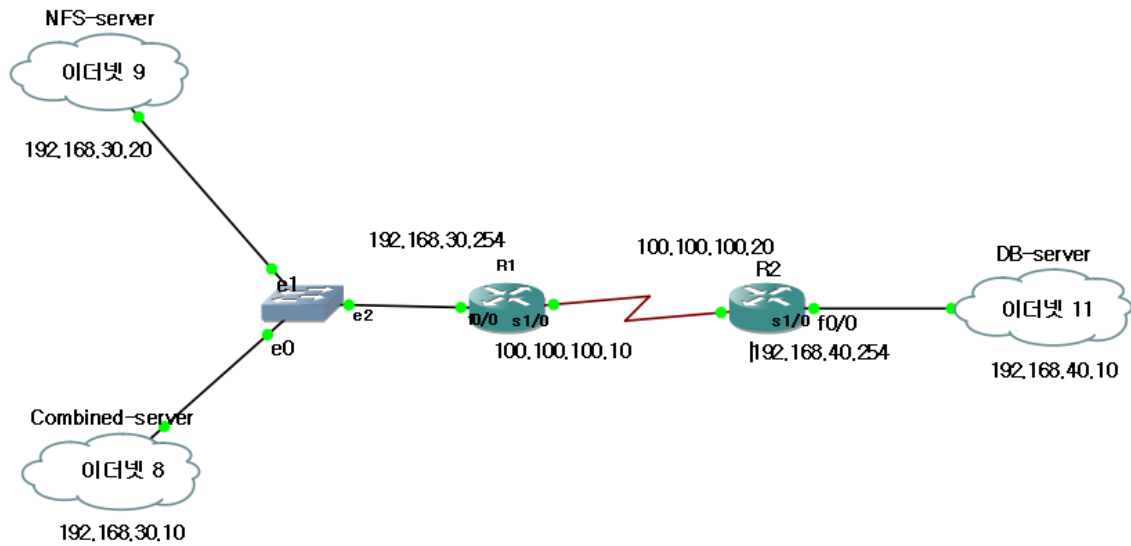
vagrant up 으로 설치

```
D:\#vagt#pt>vagrant up
```

설치 완료



## ● GNS 를 이용한 네트워크 구성



### - R1 라우터 설정

```

conf t
int f0/0
ip add 192.168.30.254 255.255.255.0
no sh
exit
int s1/0
ip add 100.100.100.10 255.255.255.0
no sh
exit
ip route 192.168.40.0 255.255.255.0 100.100.100.20
  
```

### - R2 라우터 설정

```

conf t
int f0/0
ip add 192.168.40.254 255.255.255.0
no sh
exit
int s1/0
ip add 100.100.100.20 255.255.255.0
no sh
exit
ip route 192.168.30.0 255.255.255.0 100.100.100.10
  
```



- iptables 설정

※ iptables 란? Linux 커널의 네트워크 패킷 처리와 관련된 시스템 설정을 제어하는 도구

```
iptables -I FORWARD -s 172.19.0.0/16 -d 192.168.40.10/32 -p tcp --dport 3306 -j ACCEPT
```

// 172.19.0.0/16 네트워크에서 오는 TCP 트래픽 중 목적지 포트가 3306 (MySQL 기본 포트)인 트래픽을 허용

```
iptables -I FORWARD -d 172.19.0.0/16 -s 192.168.40.10/32 -p tcp --sport 3306 -j ACCEPT
```

// 192.168.40.10 에서 172.19.0.0/16 네트워크로 가는 TCP 응답 트래픽 중 소스 포트가 3306 인 트래픽을 허용

```
iptables -t nat -A POSTROUTING -s 172.19.0.0/24 -d 192.168.40.10/32 -p tcp --dport 3306 -j SNAT --to-source 192.168.30.10
```

// 172.19.0.0/24 네트워크에서 192.168.40.10 으로 가는 MySQL 트래픽에 대해 소스 NAT 를 적용하여, 패킷의 소스 주소를 192.168.30.10 으로 변경

```
iptables -I FORWARD -s 172.20.0.0/16 -d 192.168.30.10/32 -p tcp --dport 8080 -j ACCEPT
```

// 172.20.0.0/16 네트워크에서 오는 TCP 트래픽 중 목적지 포트가 8080 (Was 포트)인 트래픽을 허용

```
iptables -I FORWARD -d 172.20.0.0/16 -s 192.168.30.10/32 -p tcp --sport 8080 -j ACCEPT
```

// 192.168.30.10 에서 172.20.0.0/16 네트워크로 가는 TCP 응답 트래픽 중 소스 포트가 8080 인 트래픽을 허용

```
iptables -t nat -A POSTROUTING -s 172.20.0.0/24 -d 192.168.40.10/32 -p tcp --dport 8080 -j SNAT --to-source 192.168.40.10
```

// 172.20.0.0/24 네트워크에서 192.168.40.10 으로 가는 웹 트래픽에 대해 소스 NAT 를 적용하여, 패킷의 소스 주소를 192.168.40.10 으로 변경



## IV. 기술 구현 방법

- Docker 컨테이너 구축

- vagrant 로 docker 설치 및 docker-compose 설치

```
sudo curl -L
```

```
"https://github.com/docker/compose/releases/download/1.29.2/dockercompos  
e-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose # 실행 권한
```

부여: 다운로드한 파일에 실행 권한을 부여

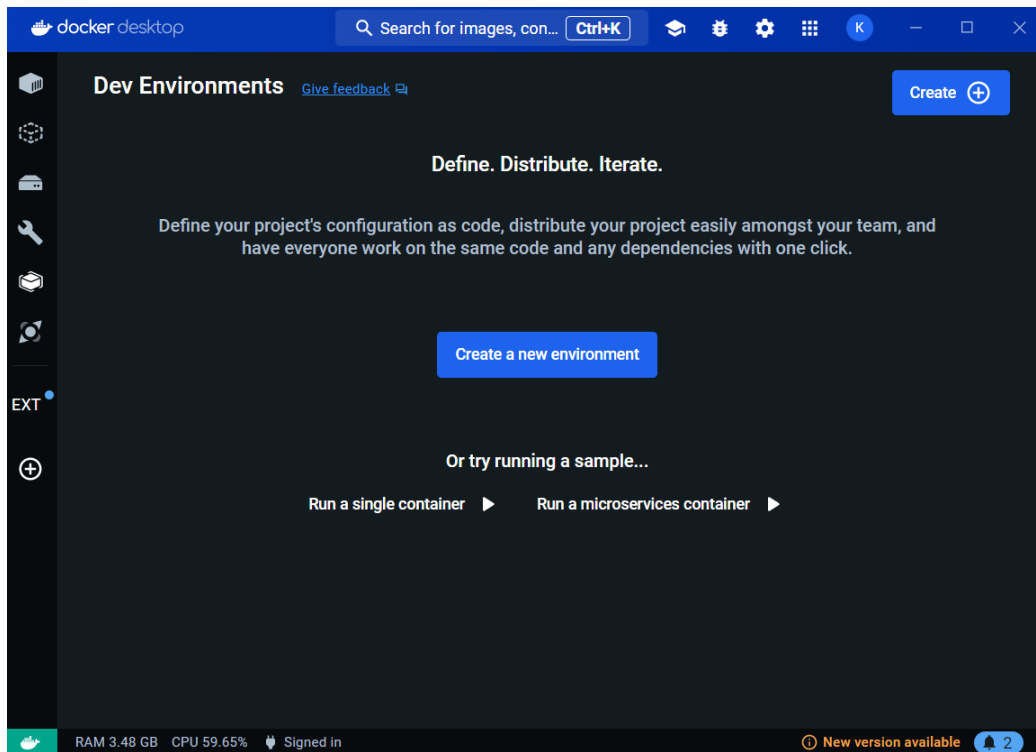
```
sudo chmod +x /usr/local/bin/docker-compose # /usr/local/bin 에 대한 실행  
경로를 설정하기 위해 심볼릭 링크를 생성
```

```
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose
```

- 보다 편리한 개발을 위해 윈도우에 docker desktop 을 설치
- wsl 을 이용할 수 있음

※ wsl 이란?

-> Windows 운영체제에서 Linux 환경을 사용할 수 있게 해주는 도구



- Web/Was/DB 구축 및 컨테이너화

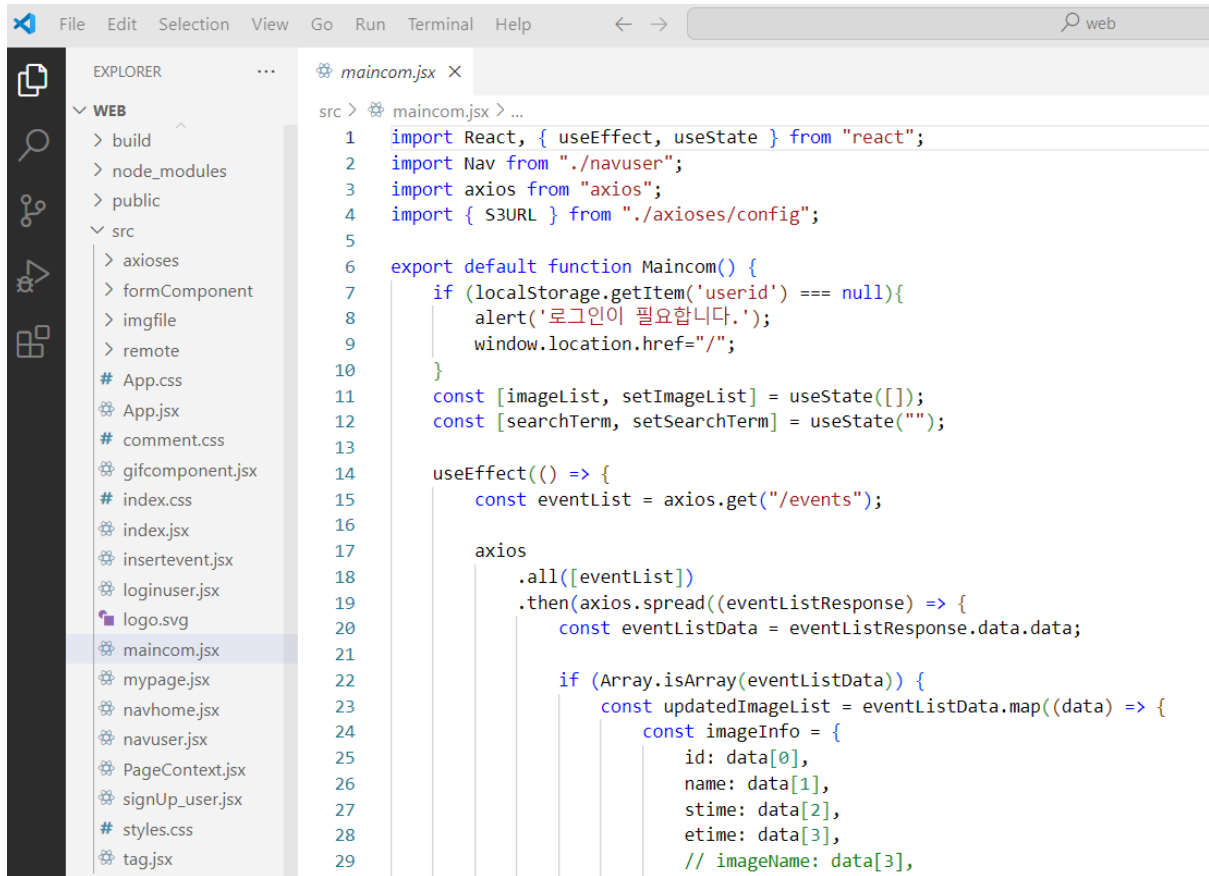
Web - react(nginx)

Was - springboot(내장 tomcat)

DB - MySQL

react 로 구성한 Web

- 파일 구성

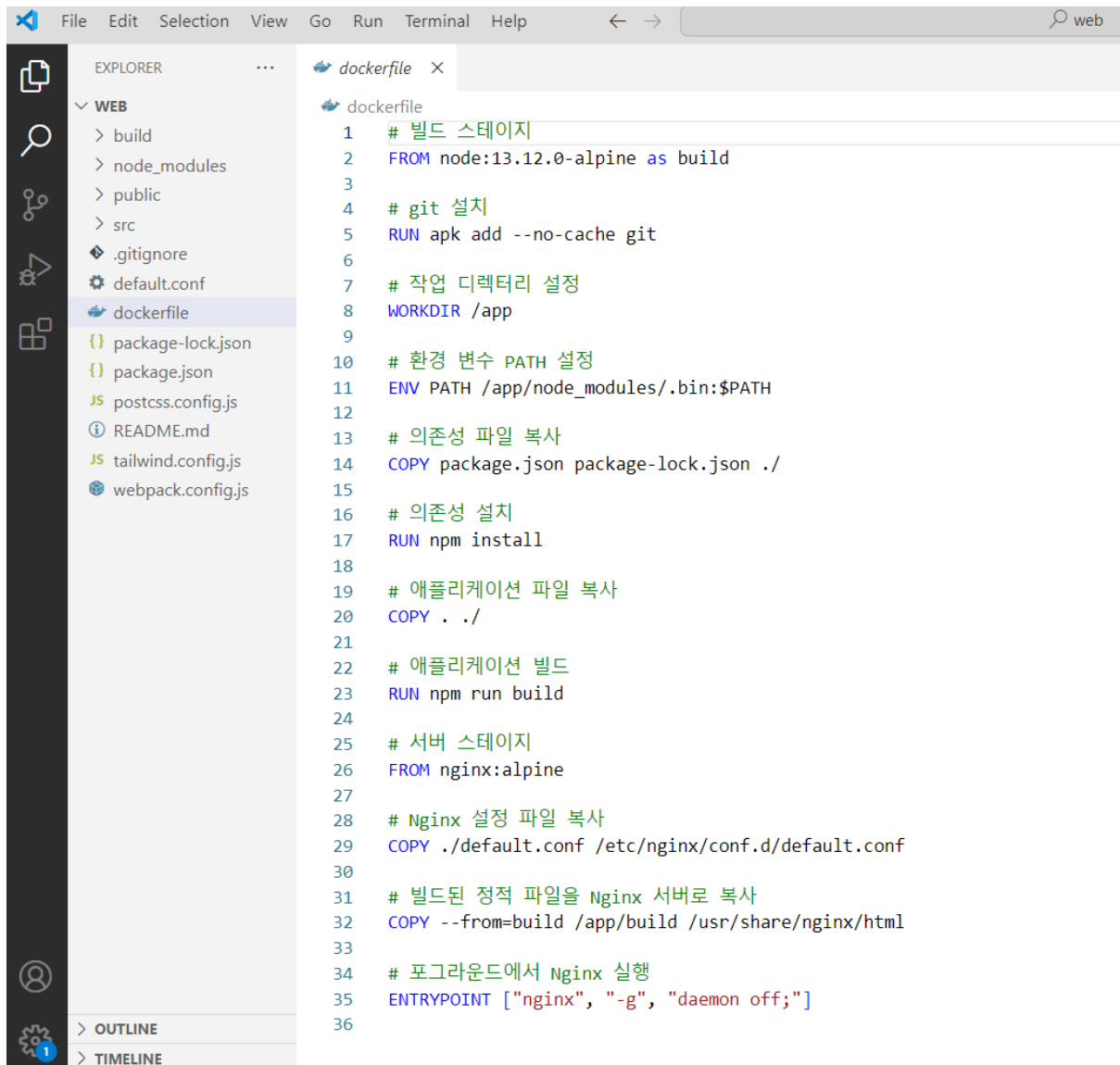


The image shows a Visual Studio Code editor window. On the left, the Explorer sidebar displays a project structure under a 'WEB' folder. The 'src' directory contains several files, with 'maincom.jsx' selected. The main editor area shows the code for 'maincom.jsx'. The code imports React, Nav, axios, and S3URL. It defines a Maincom function that checks for a user ID in localStorage and alerts the user to login if it's null. It also uses useState for imageList and searchTerm, and useEffect to fetch event data from an API endpoint '/events'.

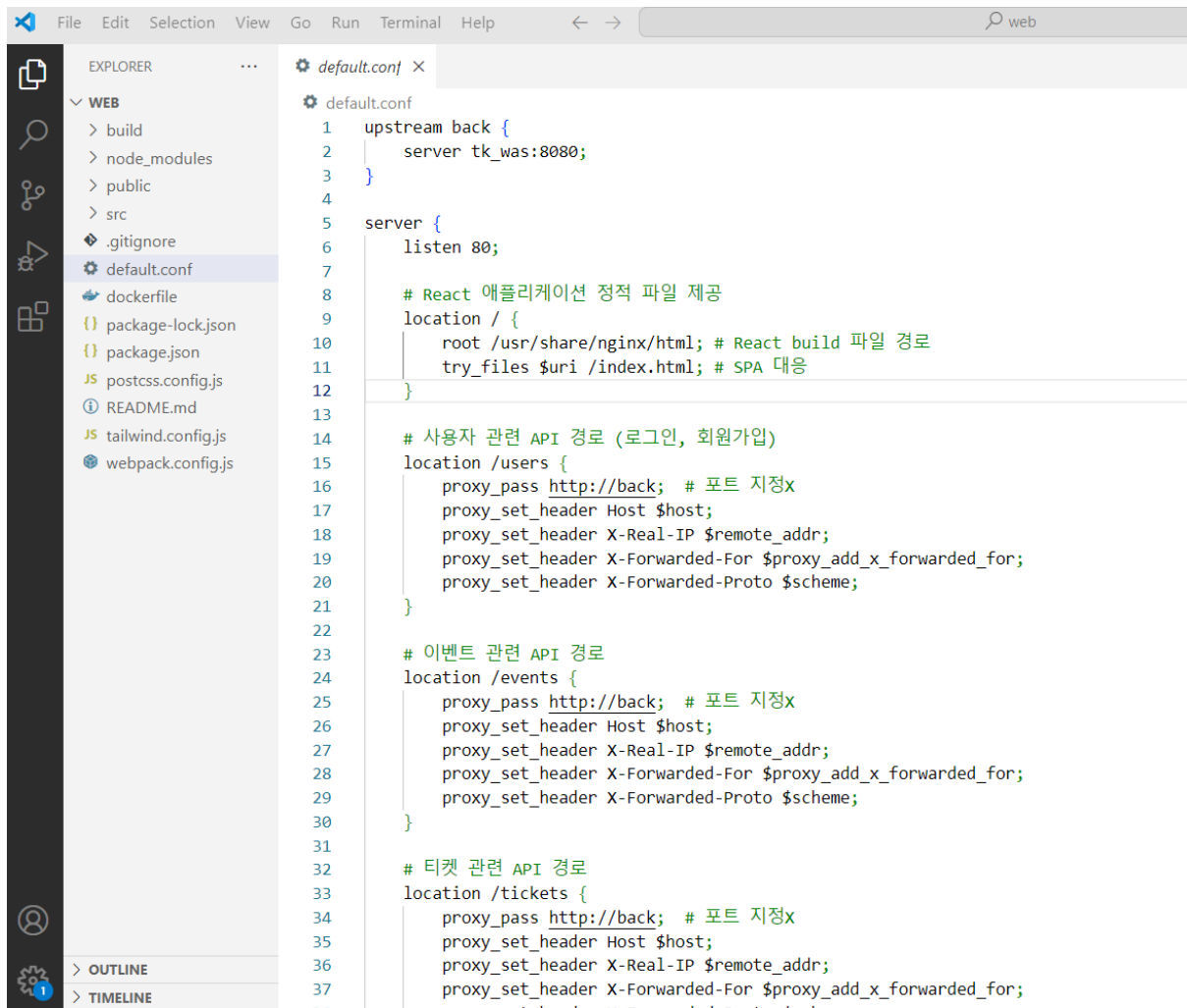
```
1 import React, { useEffect, useState } from "react";
2 import Nav from "../navuser";
3 import axios from "axios";
4 import { S3URL } from "../axioses/config";
5
6 export default function Maincom() {
7   if (localStorage.getItem('userid') === null){
8     alert('로그인이 필요합니다.');
```

```
9     window.location.href="/";
10  }
11  const [imageList, setImageList] = useState([]);
12  const [searchTerm, setSearchTerm] = useState("");
13
14  useEffect(() => {
15    const eventList = axios.get("/events");
16
17    axios
18      .all([eventList])
19      .then(axios.spread((eventListResponse) => {
20        const eventListData = eventListResponse.data.data;
21
22        if (Array.isArray(eventListData)) {
23          const updatedImageList = eventListData.map((data) => {
24            const imageInfo = {
25              id: data[0],
26              name: data[1],
27              stime: data[2],
28              etime: data[3],
29              // imageName: data[3],
```

## - dockerfile 구성

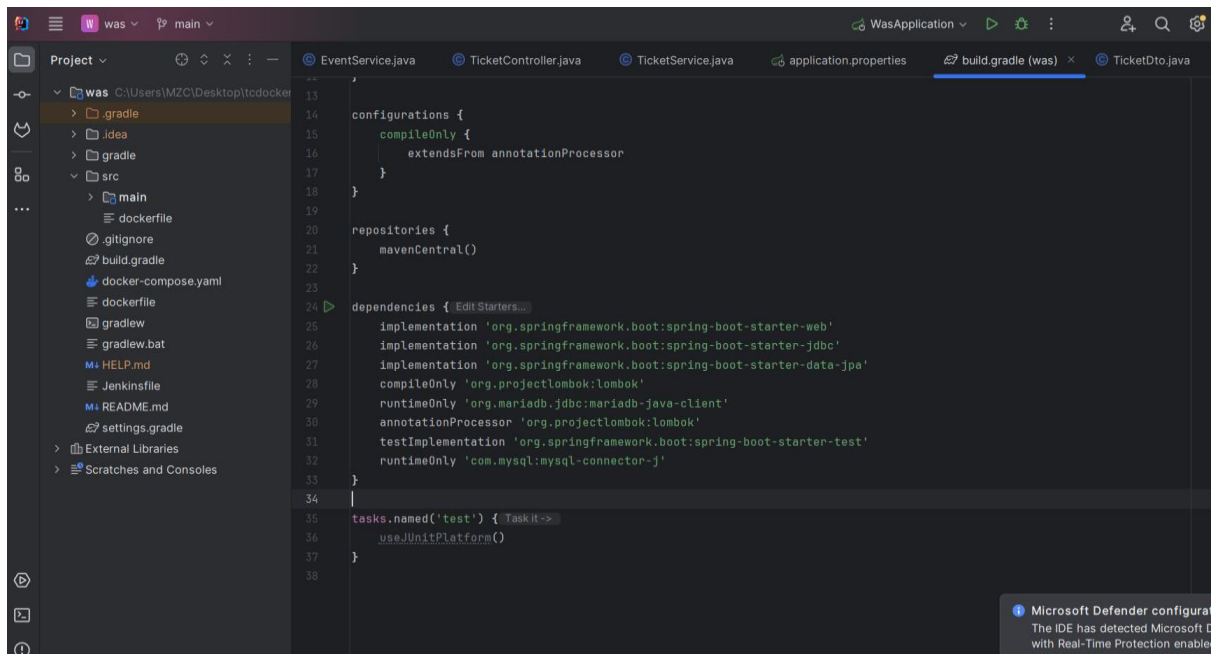


- default.conf 구성

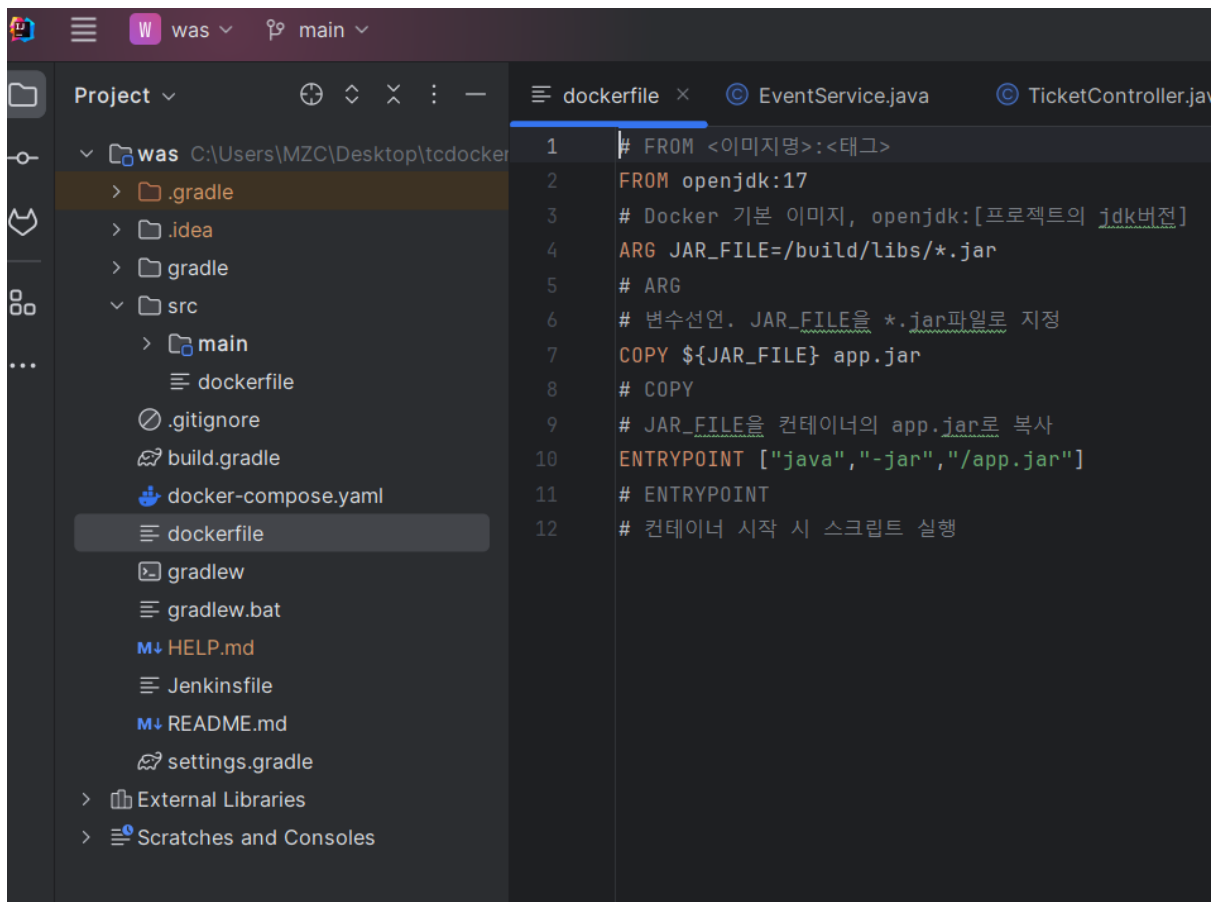


springboot 로 구성한 Was

- 파일 구성



## - dockerfile 구성



mysql 로 구성한 DB



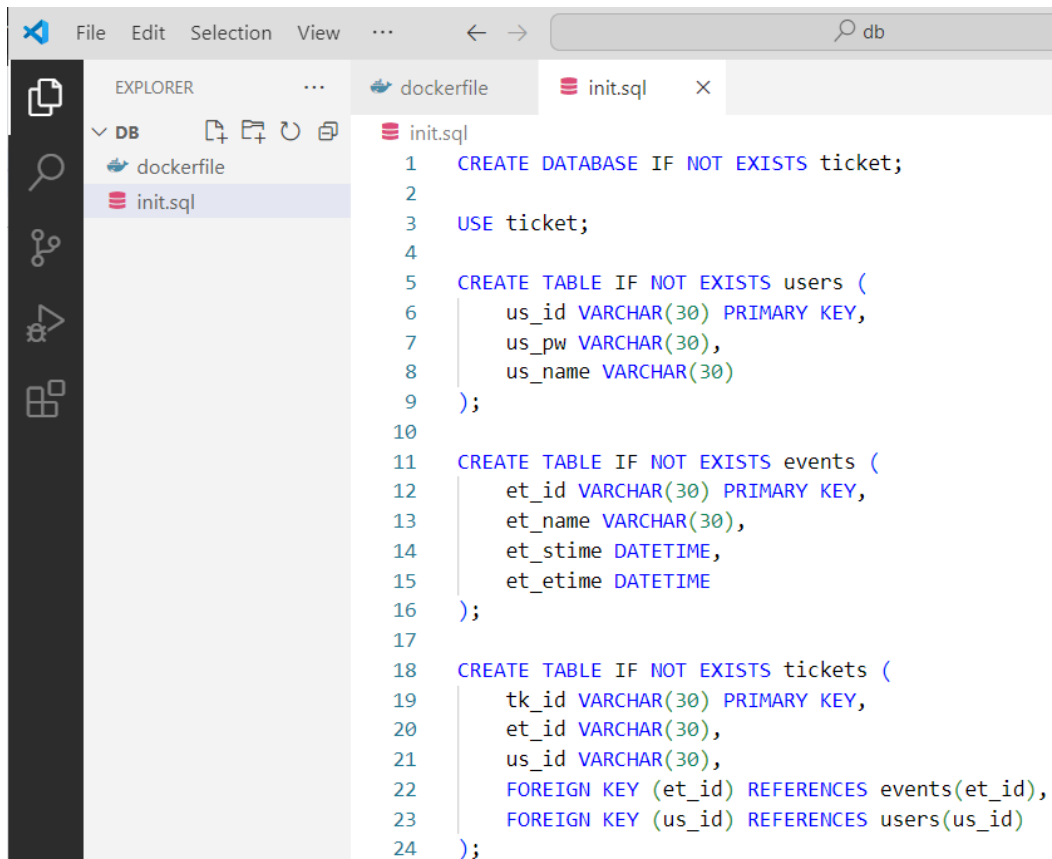
## - dockerfile 구성



The screenshot shows the VS Code interface with a Dockerfile open. The Explorer sidebar on the left shows a project named 'DB' containing 'dockerfile' and 'init.sql'. The Dockerfile content is as follows:

```
1 FROM mysql:8.0.22
2
3 ENV MYSQL_DATABASE=ticket \
4     MYSQL_ROOT_HOST=%' \
5     MYSQL_ROOT_PASSWORD=1234 \
6     TZ='Asia/Seoul'
7
8 CMD ["mysqld", "--character-set-server=utf8mb4", "--collation-server=utf8mb4_unicode"]
```

## - init.sql 구성

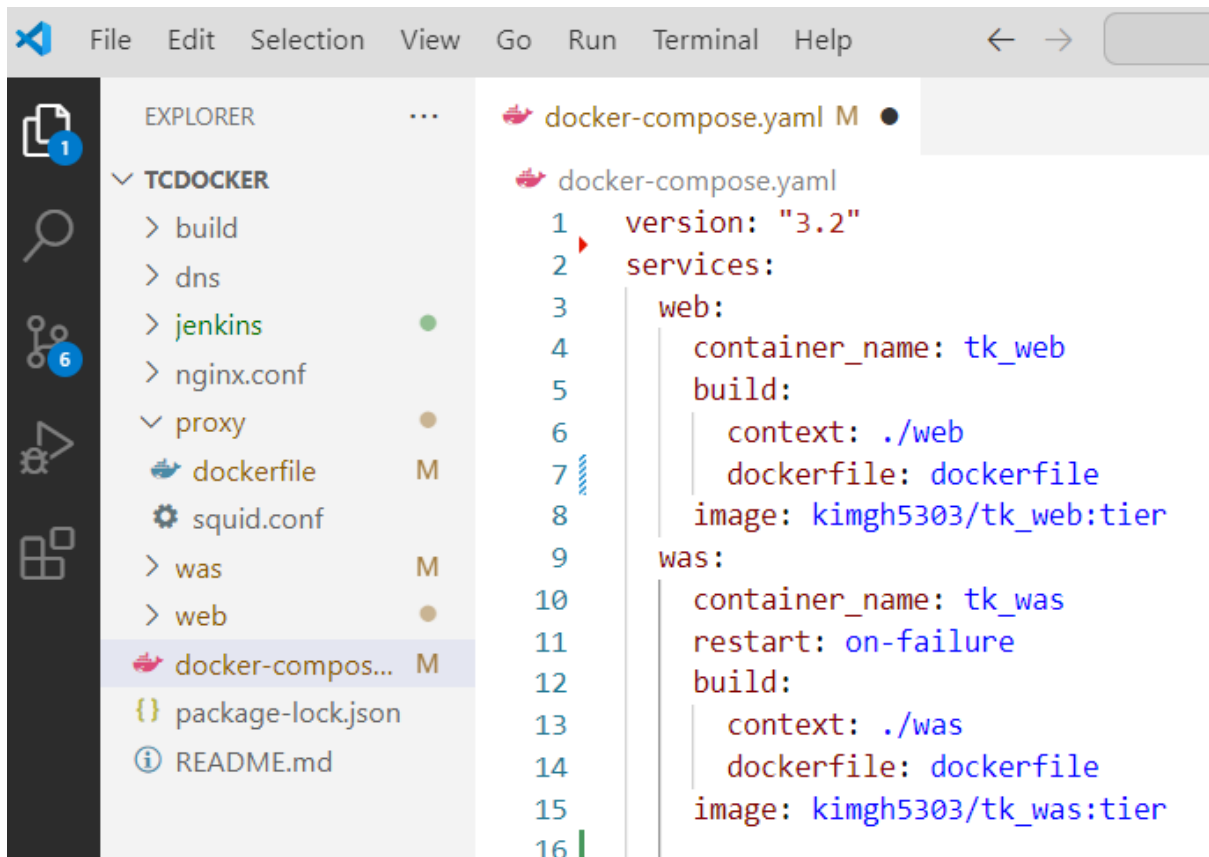


The screenshot shows the VS Code interface with an 'init.sql' file open. The Explorer sidebar on the left shows the 'DB' project with 'dockerfile' and 'init.sql'. The 'init.sql' content is as follows:

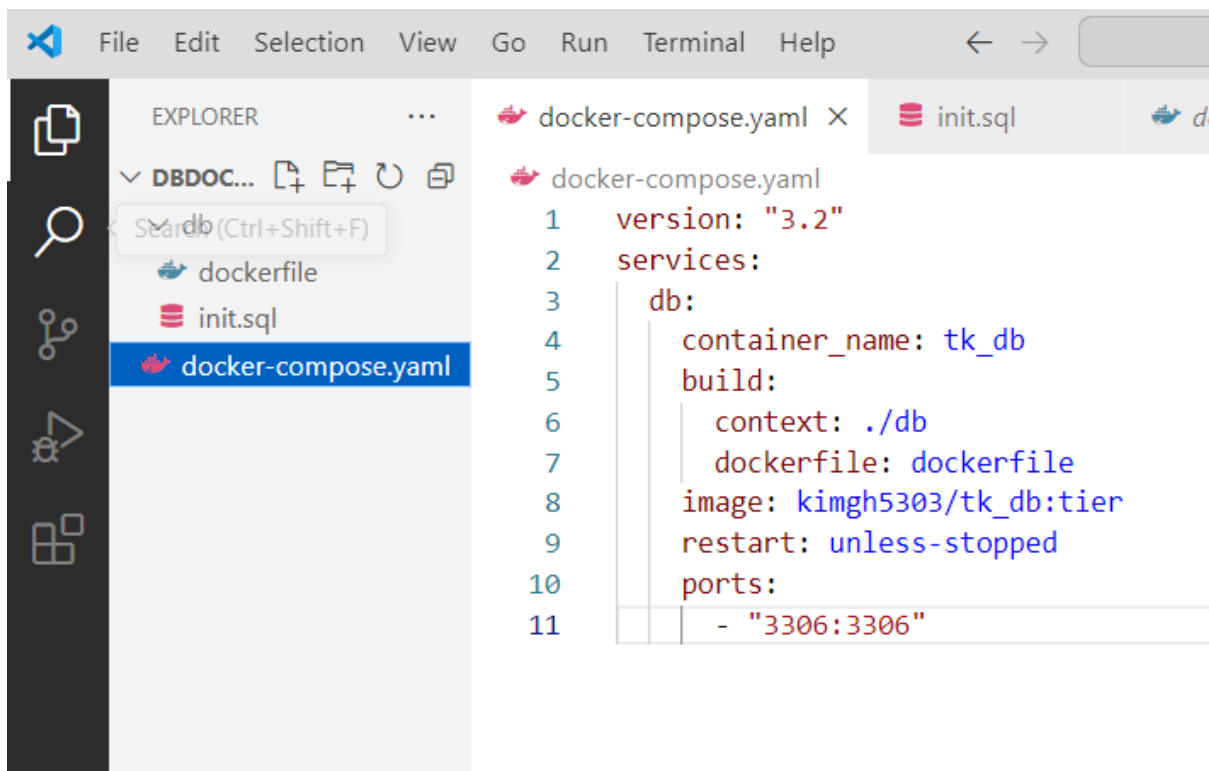
```
1 CREATE DATABASE IF NOT EXISTS ticket;
2
3 USE ticket;
4
5 CREATE TABLE IF NOT EXISTS users (
6     us_id VARCHAR(30) PRIMARY KEY,
7     us_pw VARCHAR(30),
8     us_name VARCHAR(30)
9 );
10
11 CREATE TABLE IF NOT EXISTS events (
12     et_id VARCHAR(30) PRIMARY KEY,
13     et_name VARCHAR(30),
14     et_stime DATETIME,
15     et_etime DATETIME
16 );
17
18 CREATE TABLE IF NOT EXISTS tickets (
19     tk_id VARCHAR(30) PRIMARY KEY,
20     et_id VARCHAR(30),
21     us_id VARCHAR(30),
22     FOREIGN KEY (et_id) REFERENCES events(et_id),
23     FOREIGN KEY (us_id) REFERENCES users(us_id)
24 );
```

## 전체 docker-compose 파일 구성

### - Web & Was compose 파일 구성



## - DB compose 파일 구성



## - 이미지로 빌드

# docker-compose build

PROBLEMS OUTPUT DEBUG CONSOLE TERMINAL PORTS

powershell + - [ ] [ ] ... ^ >

```
PS C:\Users\MZC\Desktop\dbdocker> docker-compose build
[+] Building 0.0s (0/0) docker:default
[+] Building 1.9s (6/6) FINISHED
=> [db internal] load build definition from dockerfile
=> => transferring dockerfile: 262B
=> [db internal] load metadata for docker.io/library/mysql:8.0.22
=> [db auth] library/mysql:pull token for registry-1.docker.io
=> [db internal] load .dockerignore
=> => transferring context: 2B
=> CACHED [db 1/1] FROM docker.io/library/mysql:8.0.22@sha256:0fd2898dc1c946b34dceacc3b80d38b1049285c1dab70df7480de62265d6213
=> [db] exporting to image
=> => exporting layers
=> => writing image sha256:864cc5e63decc1c09afdd913dd01b0dae36ebceb066913cee61cdf667f79b1cb
=> => naming to docker.io/kimgh5303/tk_db:tier
PS C:\Users\MZC\Desktop\dbdocker> █
```


```
docker:default
0.0s
0.0s
1.7s
0.0s
0.0s
0.0s
0.0s
0.0s
0.0s
0.0s
0.0s
0.0s
```

## - docker hub 에 등록 docker-compose push

```
PS C:\Users\MZC\Desktop\dbdocker> docker-compose push
```

```
[+] Pushing 12/12
✓ Pushing kimgh5303/tk_db:tier: 9a3a2a8be646 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: a9d9d14a1b5f Layer already exists
✓ Pushing kimgh5303/tk_db:tier: ab4149930225 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: 886316cd2456 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: 31c817234280 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: 98d98806c8ac Layer already exists
✓ Pushing kimgh5303/tk_db:tier: 0394a41efa73 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: c484a3b6d841 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: 6d23902c2a54 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: 74c86dff46f Layer already exists
✓ Pushing kimgh5303/tk_db:tier: ef4a33cee7a0 Layer already exists
✓ Pushing kimgh5303/tk_db:tier: cb42413394c4 Layer already exists
PS C:\Users\MZC\Desktop\dbdocker> █
```

```
1.4s
1.4s
1.3s
1.4s
1.4s
2.0s
2.2s
2.1s
2.1s
2.1s
2.8s
2.8s
```

 **dockerhub**

Explore Repositories Organizations

ctrl+K ? [ ] [ ] K

kimgh5303

▼

All Content

▼

Create repository

kimgh5303 / tk\_db

Contains: Image • Last pushed: 27 minutes ago

✖ Security unknown ☆ 0 ⬇ 8 🔓 Public

kimgh5303 / tk\_dns

Contains: Image • Last pushed: about 5 hours ago

✖ Security unknown ☆ 0 ⬇ 7 🔓 Public

kimgh5303 / tk\_jenkins

Contains: Image • Last pushed: about 12 hours ago

✖ Security unknown ☆ 0 ⬇ 3 🔓 Public

kimgh5303 / tk\_proxy

Contains: Image • Last pushed: 2 days ago

✖ Security unknown ☆ 0 ⬇ 4 🔓 Public

kimgh5303 / tk\_web

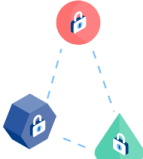
Contains: Image • Last pushed: 2 days ago

✖ Security unknown ☆ 0 ⬇ 9 🔓 Public

kimgh5303 / tk\_was


Contains: Image • Last pushed: 2 days ago

✖ Security unknown ☆ 0 ⬇ 12 🔓 Public



### Create An Organization

Create and manage users and grant access to your repositories.



## - 실행용 compose 파일 구성

```

version: "3.2"
services:
  web:
    container_name: tk_web
    restart: always
    image: kingh5303/tk_web:tier # 이전에 빌드된 이미지 사용
    ports:
      - "80:80" # 컨테이너의 80 포트를 호스트의 80 포트에 바인딩
    networks:
      - web # web 네트워크에 연결
    volumes:
      - ./web/nginx.conf:/etc/nginx/nginx.conf
      - ./web/default.conf:/etc/nginx/conf.d/default.conf
      - ./web/log:/var/log/nginx/
#-----
  was:
    container_name: tk_was
    image: kingh5303/tk_was:tier # 이전에 빌드된 이미지 사용
    restart: on-failure
    ports:
      - "8080:8080" # 컨테이너의 8080 포트를 호스트의 8080 포트에 바인딩
    environment:
      SPRING_DATASOURCE_URL: jdbc:mysql://192.168.40.10:3306/ticket?useSSL=false&allowPublicKeyRetrieval=true
      SPRING_DATASOURCE_USERNAME: root
      SPRING_DATASOURCE_PASSWORD: 1234
    networks:
      - web
      - was
    volumes:
      - /root/was/webapps:/usr/local/tomcat/webapps
      - /root/was/logs:/usr/local/tomcat/logs
      - /root/was/conf:/usr/local/tomcat/conf
#-----
  proxy:
    container_name: tk_proxy
    image: kingh5303/tk_proxy:tier
    volumes:

```

## - <docker-compose up -d>로 컨테이너 실행

```

root@ubuntu-bionic:~# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
0c4f9b38fa5b   kingh5303/tk_was:tier               "java -jar /app.jar"    4 hours ago   Up 4 hours   0.0.0.0:8080->8080/tcp, :::8080->8080/tcp
c0ef5c3412a9   kingh5303/tk_dns:tier               "named -g -c /etc/bi... 5 hours ago   Up 5 hours   0.0.0.0:53->53
3/tcp, 0.0.0.0:53->53/udp, :::53->53/tcp, :::53->53/udp
3e2ca33ab961   kingh5303/tk_web:tier               "nginx -g 'daemon of... 10 hours ago   Up 10 hours   0.0.0.0:80->80
0/tcp, :::80->80/tcp
a746087b5ca3   kingh5303/tk_jenkins:tier           "/usr/bin/tini -- /u... 11 hours ago   Up About an hour   50000/tcp, 0.
0.0.0:8081->8080/tcp, :::8081->8080/tcp
1b7548f867d5   kingh5303/tk_proxy:tier             "/sbin/entrypoint.sh... 11 hours ago   Up 11 hours   0.0.0.0:3128->
3128/tcp, :::3128->3128/tcp
root@ubuntu-bionic:~#

```

## - 웹사이트 실행

Ticket mall

로그인

원리하게 예약하고 즐기자!

모두의 티켓  
여기서 시작



- DNS 서버 구축


- Dockerfile 구성

dns >  dockerfile

```
1 FROM ubuntu:latest
2 RUN apt-get update && apt-get install -y bind9 dnsutils systemctl
3 COPY named.conf.local /etc/bind/named.conf.local
4 COPY db.tplust.kgh /etc/bind/zones/db.tplust.kgh
5 COPY db.192.168.30 /etc/bind/zones/db.192.168.30
6 CMD ["named", "-g", "-c", "/etc/bind/named.conf"]
```

```
25 | dns:
26 |   container_name: tk_dns
27 |   build:
28 |     context: ./dns
29 |     dockerfile: dockerfile
30 |   image: kimgh5303/tk_dns:tier
31 |   ports:
32 |     - "53:53/tcp"
33 |     - "53:53/udp"
```

- db.tplust.kgh

dns >  db.tplust.kgh

```
1 $TTL      604800
2 @         IN      SOA      tplust.kgh. root.tplust.kgh. (
3           |      2          ; Serial
4           | 604800        ; Refresh
5           | 86400         ; Retry
6           | 2419200       ; Expire
7           | 604800 )      ; Negative Cache TTL
8 ;
9 @         IN      NS       tplust.kgh.
10 @        IN      A        192.168.30.10
```

- db.192.168.30

```
dns > ≡ db.192.168.30  
1 $TTL      604800  
2 @         IN      SOA      30.168.192.in-addr.arpa. root.tplust.kgh. (  
3           |       |       |       |       |       |  
4           |       |       |       |       |       |   2          ; Serial  
5           |       |       |       |       |       |   604800     ; Refresh  
6           |       |       |       |       |       |   86400     ; Retry  
7           |       |       |       |       |       |  2419200    ; Expire  
8           |       |       |       |       |       |   604800 )    ; Negative Cache TTL  
9 ;  
10 @        IN      NS       tplust.kgh.  
11 10       IN      PTR      tplust.kgh.]
```

- nano /etc/resolv.conf  
nameserver 설정

```
# See man:systemd-resolved.service
# operation for /etc/resolv.conf.
```

```
#nameserver 127.0.0.53
nameserver 172.21.0.4
options edns0
```

- nslookup 으로 확인하기 (가상 머신)

```
root@ubuntu-bionic:~# nslookup tplust.kgh
Server:          172.21.0.4
Address:         172.21.0.4#53
```

Name: tplust.kgh  
Address: 192.168.30.10

```
root@ubuntu-bionic:~# nslookup 192.168.30.10
10.30.168.192.in-addr.arpa      name = tplust.kgh.
```

```
root@ubuntu-bionic:~#
```

- nslookup 으로 확인하기 (RealPC)

```

C:\Users\MZC>nslookup tplust.kgh
서버:      tplust.kgh
Address:   192.168.30.10

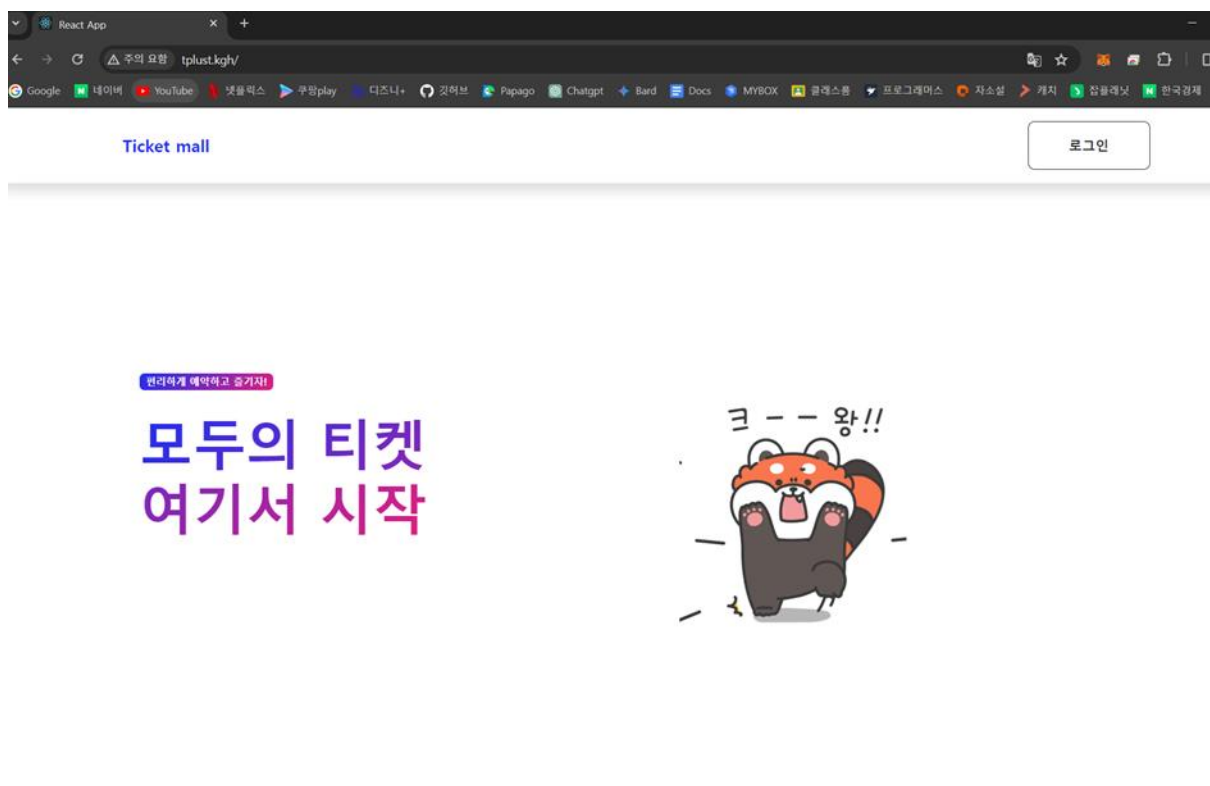
이름:      tplust.kgh
Address:   192.168.30.10

C:\Users\MZC>nslookup 192.168.30.10
서버:      tplust.kgh
Address:   192.168.30.10

이름:      tplust.kgh
Address:   192.168.30.10

```

## - 도메인 네임 접근 확인



- Jenkins 서버 구축
  - docker compose.yaml 구성

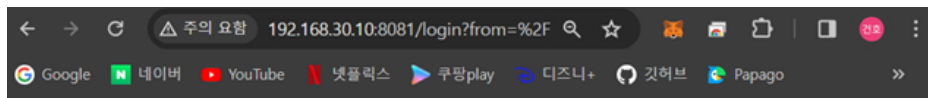


```
jenkins:
  image: jenkins/jenkins:lts
  container_name: tk_jenkins
  user: root
  restart: unless-stopped
  ports:
    - "8081:8080"
  volumes:
    - "/home/jenkins:/var/jenkins_home"
    - "/var/run/docker.sock:/var/run/docker.sock"
```

## - Dockerfile 구성

```
jenkins > dockerfile
1 | Jenkins LTS 이미지를 베이스 이미지로 사용
2 FROM jenkins/jenkins:lts
3
4 # root로 사용자 변경
5 USER root
6
7 # 필수 패키지 설치, Docker 공식 GPG 키 추가 및 저장소 설정
8 RUN apt-get update && \
9     apt-get install -y \
10     apt-transport-https \
11     ca-certificates \
12     curl \
13     gnupg2 \
14     software-properties-common && \
15     curl -fsSL https://download.docker.com/linux/debian/gpg | apt-key add - && \
16     add-apt-repository \
17     "deb [arch=amd64] https://download.docker.com/linux/debian \
18     $(lsb_release -cs) stable" && \
19     apt-get update && \
20     apt-get install -y docker-ce docker-ce-cli containerd.io
21
22 # Docker Compose 설치
23 RUN curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" \
24     -o /usr/local/bin/docker-compose && \
25     chmod +x /usr/local/bin/docker-compose
26
27 # 다시 Jenkins 사용자로 변경
28 USER jenkins
29
```

## - 접속 확인



## Sign in to Jenkins

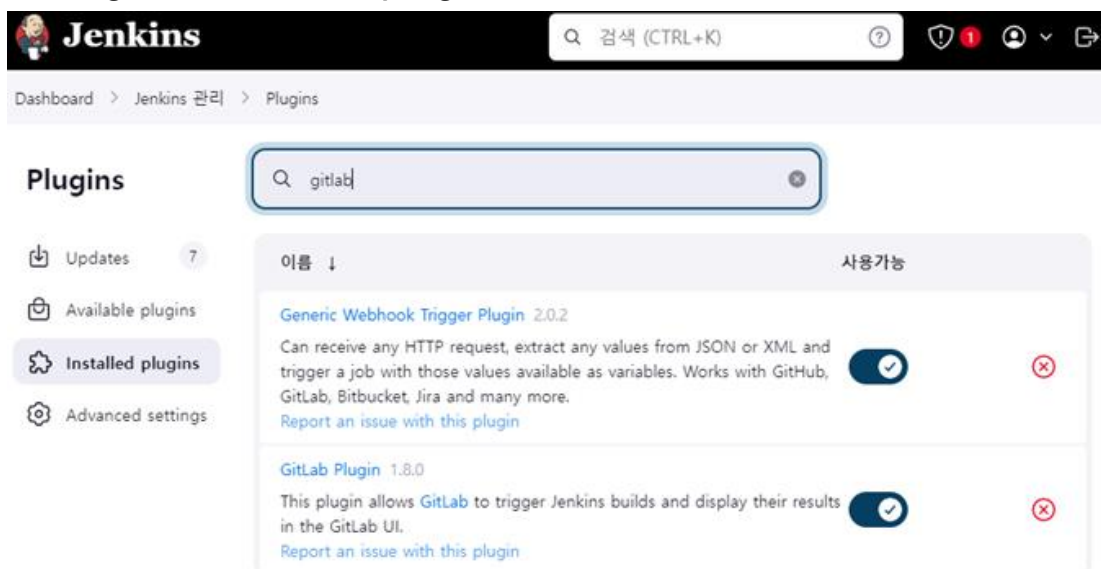
사용자 이름

비밀번호

☐ 로그인 상태 유지

로그인

- gitlab 서버 구축
- gitlab webhook plugin 설치



- Jenkins 의 인터넷 연결 문제로 인해 plugin 설치 불가

164

```

• Checking internet connectivity
• Checking update center connectivity
• java.net.ConnectException: Connection refused
  at java.base/sun.nio.ch.Net.pollConnect(Native Method)
  at java.base/sun.nio.ch.Net.pollConnectNow(Unknown Source)
  at java.base/sun.nio.ch.NioSocketImpl.timedFinishConnect(Unknown Source)
  at java.base/sun.nio.ch.NioSocketImpl.connect(Unknown Source)
  at java.base/java.net.Socket.connect(Unknown Source)
  at java.base/sun.net.NetworkClient.doConnect(Unknown Source)
  at java.base/sun.net.www.http.HttpClient.openServer(Unknown Source)
  at java.base/sun.net.www.http.HttpClient$1.run(Unknown Source)
  at java.base/sun.net.www.http.HttpClient$1.run(Unknown Source)
  at java.base/java.security.AccessController.doPrivileged(Unknown Source)
  at java.base/sun.net.www.http.HttpClient.privilegedOpenServer(Unknown Source)
  at java.base/sun.net.www.http.HttpClient.openServer(Unknown Source)
  at java.base/sun.net.www.protocol.https.HttpsClient.<init>(Unknown Source)
  at java.base/sun.net.www.protocol.https.HttpsClient.New(Unknown Source)
  at java.base/sun.net.www.protocol.https.AbstractDelegateHttpsURLConnection.getNewHttpClient(Unknown Source)
  at java.base/sun.net.www.protocol.http.HttpURLConnection.plainConnect0(Unknown Source)
  at java.base/sun.net.www.protocol.http.HttpURLConnection.plainConnect(Unknown Source)
  at java.base/sun.net.www.protocol.https.AbstractDelegateHttpsURLConnection.connect(Unknown Source)
  at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream0(Unknown Source)
  at java.base/sun.net.www.protocol.http.HttpURLConnection.getInputStream(Unknown Source)
  at java.base/java.net.HttpURLConnection.getResponseCode(Unknown Source)
  at java.base/sun.net.www.protocol.https.HttpsURLConnectionImpl.getResponseCode(Unknown Source)
  at hudson.model.UpdateCenter$UpdateCenterConfiguration.testConnection(UpdateCenter.java:1452)
  at hudson.model.UpdateCenter$UpdateCenterConfiguration.checkUpdateCenter(UpdateCenter.java:1223)
  at hudson.model.UpdateCenter$ConnectionCheckJob.run(UpdateCenter.java:1686)
  at java.base/java.util.concurrent.Executors$RunnableAdapter.call(Unknown Source)
  at java.base/java.util.concurrent.FutureTask.run(Unknown Source)
  at hudson.remoting.AtMostOneThreadExecutor$Worker.run(AtMostOneThreadExecutor.java:121)
  at java.base/java.lang.Thread.run(Unknown Source)

```

OWASP Markup Formatter  Failure - Details

Loading plugin extensions  Success

- Forward Proxy 로 서버와 인터넷 간 연결 수행해야 함
- Squid 로 Forward Proxy 를 만들어 줌

❖ Forward Proxy : 클라이언트를 대신해서 간접적으로 인터넷에 접속할 수 있는 중계기 역할을 하는 서버

❖ Squid : HTTP, HTTPS, FTP 등을 지원하는 웹용 캐싱 프록시

proxy:

```

container_name: tk_proxy
image: kimggh5303/tk_proxy:tier
volumes:
  - squid_data:/var/spool/squid
ports:
  - "3128:3128"
restart: always
networks:
  - root_default

```

proxy > 🐳 dockerfile

```
1  # Squid 이미지를 기반으로 합니다.
2  FROM sameersbn/squid:3.5.27-2
3
4  # 호스트의 squid.conf 파일을 이미지 내부로 복사합니다.
5  COPY squid.conf /etc/squid/squid.conf
6
7  # 컨테이너 실행 시 Squid를 시작합니다.
8  CMD ["squid", "-NYCd", "1"]
```

proxy > ⚙️ squid.conf

```
1  http_port 3128
2
3  # 모든 요청을 허용합니다
4  http_access allow all
5
6  # DNS 설정 - 필요에 따라 변경할 수 있습니다
7  dns_nameservers 8.8.8.8 8.8.4.4
8
9  # 로그 파일 설정
10 access_log /var/log/squid/access.log squid
11 cache_log /var/log/squid/cache.log
12
13 # 캐시 디렉토리 설정 - 필요에 따라 변경할 수 있습니다
14 cache_dir ufs /var/spool/squid 100 16 256
```

- mkdir -p /etc/systemd/system/docker.service.d
- nano /etc/systemd/system/docker.service.d/http-proxy.conf

### [[Service]

```
Environment="HTTP_PROXY=http://tk_proxy:3128"
Environment="HTTPS_PROXY=http://tk_proxy:3128"
Environment="NO_PROXY=localhost,127.0.0.1"
```

- sudo systemctl daemon-reload
- sudo systemctl restart docker
- nano /etc/hosts

```
127.0.0.1      localhost
172.21.0.2     tk_proxy
# The following lines are desirable for IPv6 capable hosts
::1           ip6-localhost   ip6-loopback
fe00::0       ip6-localnet
ff00::0       ip6-mcastprefix
ff02::1       ip6-allnodes
ff02::2       ip6-allrouters
ff02::3       ip6-allhosts
127.0.1.1     ubuntu-bionic    ubuntu-bionic
```

- Jenkins proxy 설정

## HTTP Proxy Configuration

Setup ^

 Edited

서버 ?

192.168.30.10

포트 ?

3128

사용자명 ?

암호



Concealed

Change Password

Dashboard > Jenkins 관리 > Plugins



Installed plugins



Advanced settings



Download progress

## Download progress

준비

- Checking internet connectivity
- Checking update center connectivity
- Success

Resource Disposer

✓ 성공

Workspace Cleanup

✓ 성공

Loading plugin extensions

✓ Success

Timestampers

✓ 성공

Loading plugin extensions

✓ Success

→ [메인 페이지로 돌아가기](#)

(설치된 플러그인을 바로 이용하실 수 있습니다.)

→ ☐ 설치가 끝나고 실행중인 작업이 없으면 Jenkins 재시작.

정상적인 연결 가능

## ● Jenkins-gitlab 연결

- Jenkins 관리 -> system 설정에서 gitlab 과 연결
- gitlab 에서 GitLab API token 발급 후 Add 를 클릭한 다음 token 값 입력

### GitLab

☒ Enable authentication for '/project' end-point ?

GitLab connections

Connection name ?  
A name for the connection

gitlab

GitLab host URL ?  
The complete URL to the GitLab server (e.g. http://gitlab.mydomain.com)

http://192.168.0.7

Credentials ?  
API Token for accessing GitLab

GitLab API token

+ Add ▾

고급 ▾

Success

Test Connection

## ● Jenkins gitlab repository 연결

- 새로운 item → pipeline

Jenkins

검색 (CTRL+K)

Dashboard > All >

Enter an item name

> This field cannot be empty, please enter a valid name

Freestyle project

Classic, general-purpose job type that checks out from up to one SCM, executes build steps serially, followed by post-build steps like archiving artifacts and sending email notifications.

Pipeline

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

Multi-configuration project

다양한 환경에서의 테스트, 플러그인 특성 빌드, 기타 동등 처럼 다수의 서로 다른 환경설정이 필요한 프로젝트에 적합함.

Folder

Creates a container that stores nested items in it. Useful for grouping things together. Unlike view, which is just a filter, a folder creates a separate namespace, so you can have multiple things of the same name as long as they are in different folders.

Multibranch Pipeline

Creates a set of Pipeline projects according to detected branches in one SCM repository.

Organization Folder

Set of multibranch project subfolders by scanning for repositories.

OK

## - gitlab webhook url 설정 체크

Dashboard > kimgh-test > Configuration

### Configure

- General
- Advanced Project Options
- Pipeline

☐ Build when a change is pushed to GitLab. GitLab webhook URL: http://192.168.30.10:8081/project/kimgh-test ?

☐ Build periodically ?

☒ Build when a change is pushed to GitLab. GitLab webhook URL: http://192.168.30.10:8081/project/kimgh-test ?

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

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events



- Secret token 발급 → gitlab webhook 연결에 이용

고급 ^

☒ Enable [ci-skip] ?

☒ Ignore WIP Merge Requests ?

Labels that launch a build if they are added (comma-separated) ?

☒ Set build description to build cause (eg. Merge request or Git Push) ?

☐ Build on successful pipeline events

Pending build name for pipeline ?

☐ Cancel pending merge request builds on update ?

Allowed branches

☒ Allow all branches to trigger this job ?

☐ Filter branches by name ?

☐ Filter branches by regex ?

☐ Filter merge request by label

Secret token ?

Generate

## - Pipeline 설정 입력

### Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

http://192.168.0.7/kimgh5303/docker.git

❗ Failed to connect to repository : Command "git ls-remote -h -- http://192.168.0.7/kimgh5303/docker.git HEAD" returned status code 128:  
stdout:  
stderr: remote: HTTP Basic: Access denied. The provided password or token is incorrect or your account has 2FA enabled and you must use a personal access token instead of a password. See [http://192.168.0.7/help/topics/git/troubleshooting\\_git#error-on-git-fetch-http-basic-access-denied](http://192.168.0.7/help/topics/git/troubleshooting_git#error-on-git-fetch-http-basic-access-denied)  
fatal: Authentication failed for 'http://192.168.0.7/kimgh5303/docker.git/'

Credentials ?

- none -

+ Add ▾

다음과 같이 뜨면 credentials 추가하면 됨

- credentials 항목 안에서 + Add
- Add 에서 gitlab id, password 입력

## Pipeline

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

http://192.168.0.7/kimgh5303/tk\_was.git

Credentials ?

kimgh5303/\*\*\*\*\* (gitlab rep)

+ Add

그룹

성공적으로 연결할 수 있는 상태

- Branch 이름 변경 master → main
- github/gitlab 전부 master 라는 용어 문제로 main 으로 설정 돼 있음

Branches to build ?

Branch Specifier (blank for 'any') ?

\*/main

Add Branch

Repository browser ?

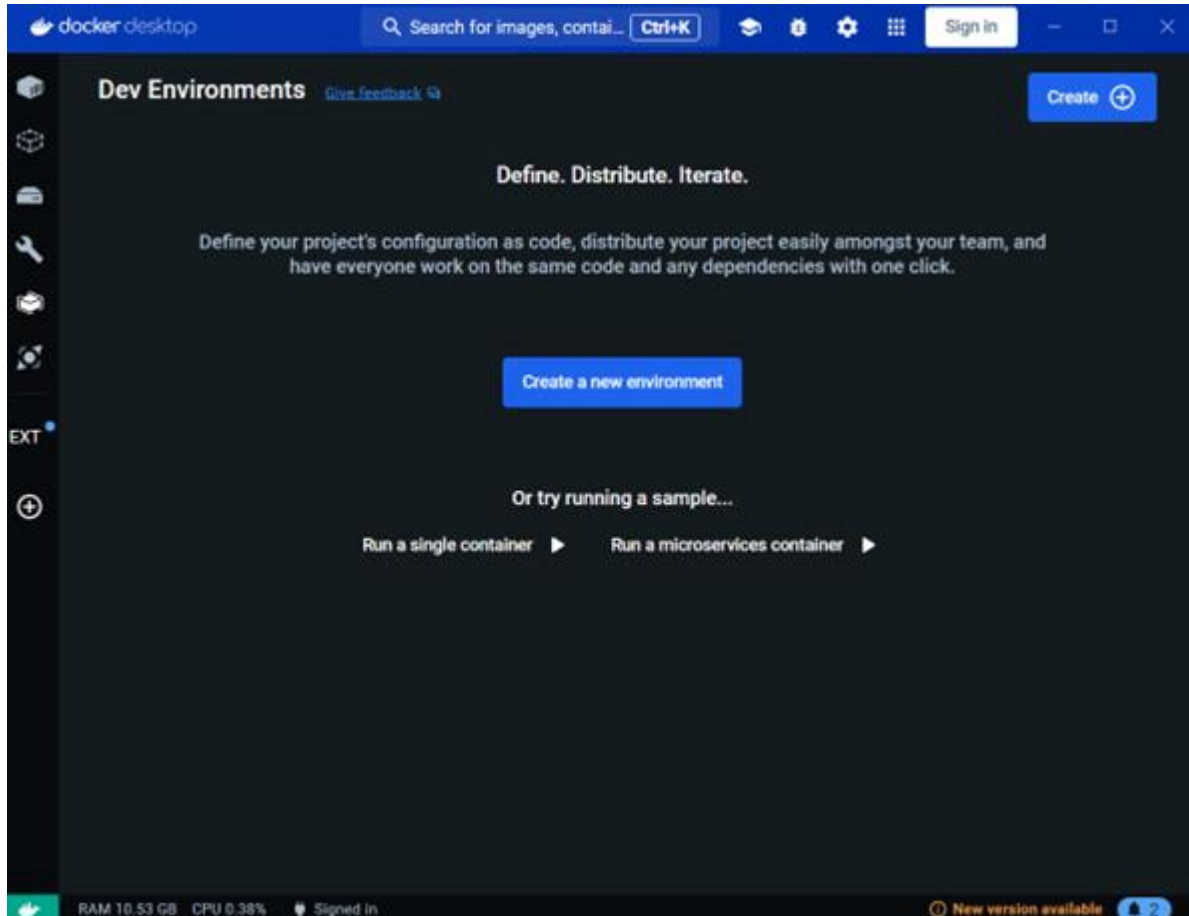
(자동)

Additional Behaviours

Add

- gitlab 구성

- docker desktop 설치
- wsl(Windows Subsystem for Linux)을 이용하여 윈도우에 gitlab 인스턴스 생성하기 위함



- gitlab 인스턴스 생성

- docker run --detach --publish 443:443 --publish 80:80 --publish 22:22 --name gitlab --restart always gitlab/gitlab-ce:latest

```
PS C:\Users\MZC> docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
eccdce07060b   gitlab/gitlab-ce:latest             "/assets/wrapper"       53 minutes ago Up 3 minutes (healthy)  0.0.0.0:22->22/tcp, 0
.0.0.0:80->80/tcp, 0.0.0.0:443->443/tcp
PS C:\Users\MZC>
```


- 컨테이너 접속 후 gitlab 구성 파일 수정

```
##: NOTE: During installation/upgrades, the value of the
##! EXTERNAL_URL will be used to populate/replace this v
##! On AWS EC2 instances, we also attempt to fetch the p
##! address from AWS. For more details, see:
##! https://docs.aws.amazon.com/AWSEC2/latest/UserGuide/
# external_url 'GENERATED_EXTERNAL_URL'
external_url 'http://192.168.0.7/'
gitlab_rails['gitlab_ssh_host']='192.168.0.7'

## Roles for multi-instance GitLab
##! The default is to have no roles enabled, which resul
##! Options:
##!   redis_sentinel_role redis_master_role redis_replic
##!   postgres_role consul_role application_role monitor
```

// external\_url 'http://192.168.0.7/' -> gitlab 서버가 사용할 공개 url 설정

// gitlab\_rails['gitlab\_ssh\_host']='192.168.0.7' -> gitlab ssh 접속 시 사용할  
호스트 ip 주소, 도메인 정의



**GitLab Community Edition**

Username or primary email

Password

[Forgot your password?](#)

☐ Remember me

**Sign in**

Don't have an account yet? [Register now](#)

- gitlab 서버 생성

## - access token 발급

🏠 User Settings / Access Tokens

Search settings

### Personal Access Tokens

You can generate a personal access token for each application you use that needs access to the GitLab API. You can also use personal access tokens to authenticate against Git over HTTP. They are the only accepted password when you have Two-Factor Authentication (2FA) enabled.

Active personal access tokens <span>🗨 1</span>						<button>Add new token</button>
Token name	Scopes	Created	Last Used <span>?</span>	Expires	Action	
gitlab_token	api, read_api, read_user, create_runner, k8s_proxy, read_repository, write_repository, ai_features	Mar 16, 2024	34 minutes ago	in 4 weeks		

## \*따로 기록해둘 것\*

## - repository 생성

🏠 Your work / Projects / New project / Create blank project



### Create blank project

Create a blank project to store your files, plan your work, and collaborate on code, among other things.

#### Project name

My awesome project

Must start with a lowercase or uppercase letter, digit, emoji, or underscore. Can also contain dots, pluses, dashes, or spaces.

#### Project URL

http://192.168.0.7/kimgh5303/

#### Project slug

my-awesome-project

#### Visibility Level ?



Private

Project access must be granted explicitly to each user. If this project is part of a group, access is granted to members of the group.



Internal

The project can be accessed by any logged in user except external users.



Public

The project can be accessed without any authentication.

#### Project Configuration



Initialize repository with a README

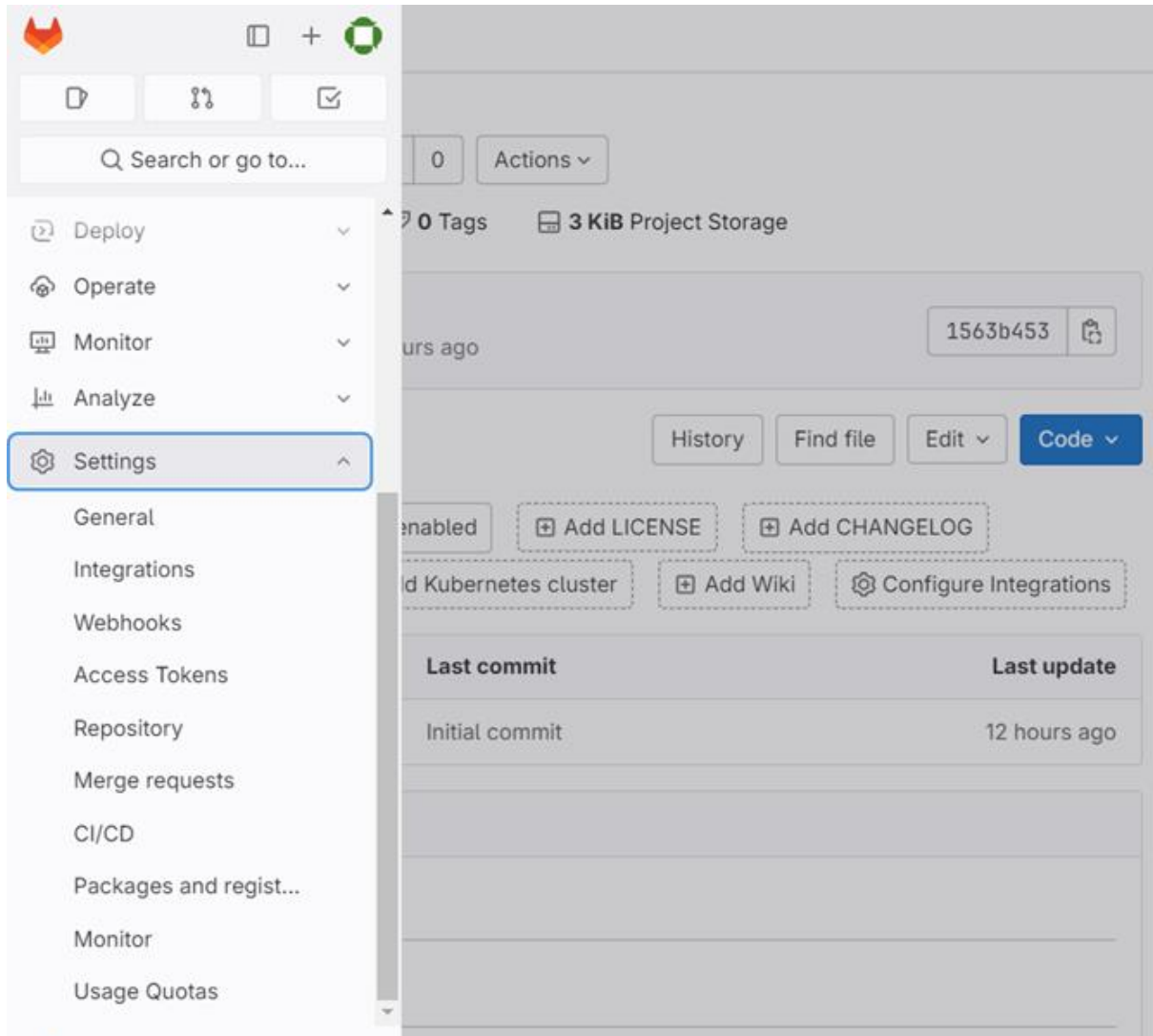
Allows you to immediately clone this project's repository. Skip this if you plan to push up an existing repository.



Enable Static Application Security Testing (SAST)

- 자동 빌드를 위한 **webhook** 설정

- ❖ **webhook** 이란? 데이터가 변경되었을 때 실시간으로 알림을 받을 수 있는 기능



Q Search page

## Webhook

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

### URL

http://192.168.30.10:8081/project/kimgh-test

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

☒ Show full URL

☐ Mask portions of URL

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

### Name (optional)

### Description (optional)

### Secret token

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

### Trigger

☒ Push events

☒ All branches

☐ Wildcard pattern

☐ Regular expression

☐ Tag push events

A new tag is pushed to the repository.

☐ Comments

A comment is added to an issue or merge request.

- push event 클릭했을 때 테스트 진행



Hook executed successfully: HTTP 200

Search page

## Webhooks

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

Project Hooks 1

Add new webhook

http://192.168.30.10:8081/project/kimgh-test

Push events SSL Verification: enabled

Test

Edit

Delete

Search page

## Webhooks

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

Project Hooks 1

Add new webhook

http://192.168.30.10:8081/project/kimgh-test

Push events SSL Verification: enabled

Test

Edit

Delete

Push events

Tag push events

Issues events

Confidential issues events

Comments

성공적으로 실행됨

- Url is blocked: Requests to the local network are not allowed

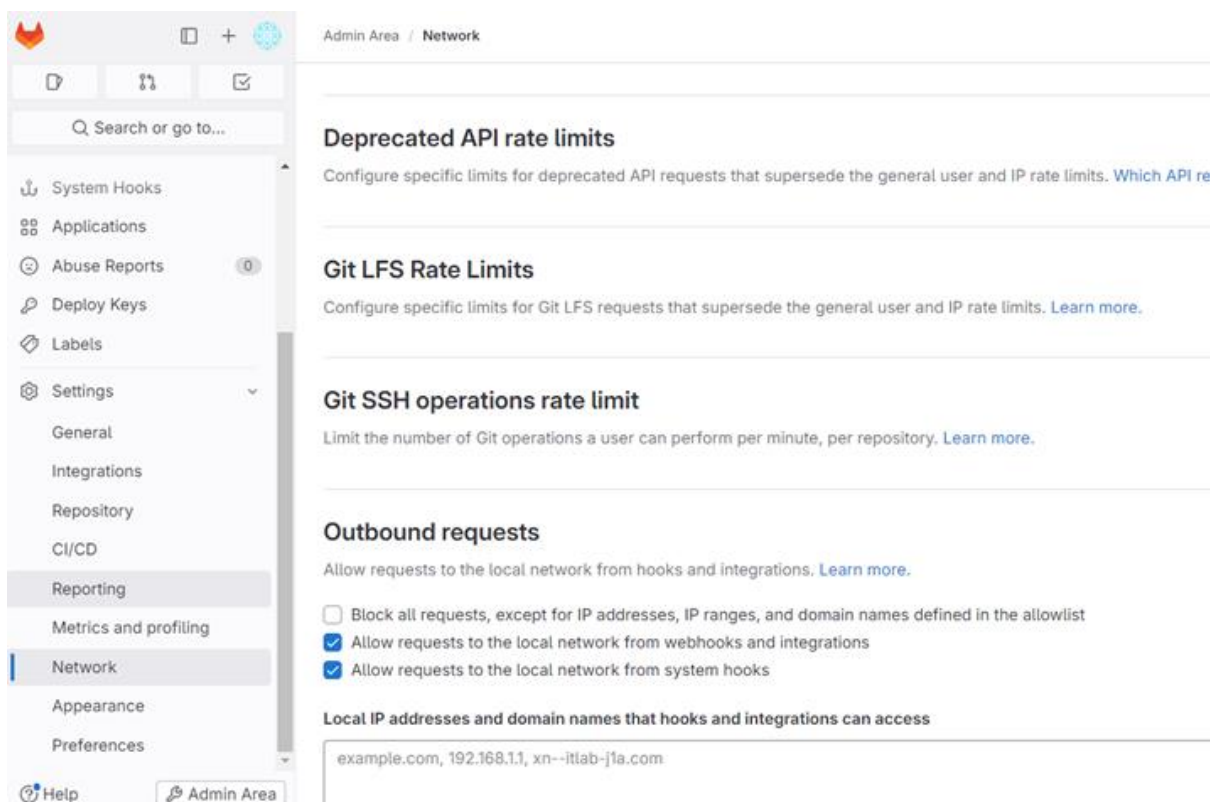
Url is blocked: Requests to the local network are not allowed

Integrations

URL

- Gitlab webhook 이용 시 보안상의 이유로 로컬 네트워크 주소로의 요청을 차단함
- 관리자 계정에서 허용해야 함. **Default** 설정은 차단중임
- root 로 로그인
- root 비밀번호 조회

```
PS C:\Users\MZC> docker exec -it gitlab grep 'Password:' /etc/gitlab/initial_root_password
Password: bzNxJn3A9uTwjBurcq2/6ytS0jKb9dG3G0UUB5oI6hc=
PS C:\Users\MZC> █
```



- 파이프라인 자동화 코드
  - 다음과 같이 루트 디렉터리에 jenkinsfile 과 dockerfile, docker-compose.yaml 작성
  - dockerfile

was >  dockerfile

```
1  # FROM <이미지명>:<태그>
2  FROM openjdk:17
3  # Docker 기본 이미지, openjdk:[프로젝트의 jdk버전]
4  ARG JAR_FILE=/build/libs/*.jar
5  # ARG
6  # 변수선언. JAR_FILE을 *.jar파일로 지정
7  COPY ${JAR_FILE} app.jar
8  # COPY
9  # JAR_FILE을 컨테이너의 app.jar로 복사
10 ENTRYPOINT ["java","-jar","/app.jar"]
11 # ENTRYPOINT
12 # 컨테이너 시작 시 스크립트 실행
```

## - docker-compose.yaml

was >  docker-compose.yaml


```

1  version: "3.2"
2  services:
3    was:
4      container_name: tk_was
5      restart: on-failure
6      build:
7        context: .
8        dockerfile: dockerfile
9      image: kimgh5303/tk_was:tier
10     ports:
11       - "8080:8080"
12     environment:
13       SPRING_DATASOURCE_URL: jdbc:mysql://192.168.40.10:3306/ticket?useS
14       SPRING_DATASOURCE_USERNAME: "root"
15       SPRING_DATASOURCE_PASSWORD: "1234"
16     networks:
17       - web
18       - was
19 networks:
20   web:
21     external: true
22   was:
23     external: true

```

## - Jenkinsfile (J 는 무조건 대문자 고정)

```

was >  Jenkinsfile
1  pipeline {
2    agent any
3    stages {
4      // 실행 파일 빌드
5      stage('Build') {
6        steps {
7          sh 'chmod +x ./gradlew'
8          sh './gradlew build -x test'
9        }
10     }
11     // 빌드 파일 테스트
12     stage('Test') {
13       steps {
14         sh './gradlew test'
15       }
16     }
17     // 컨테이너 중지
18     stage('Prepare Docker Environment') {
19       steps {
20         script {
21           // "tk_was" 컨테이너가 이미 존재하는 경우, 이를 중지하고 제거
22           sh 'docker ps -q --filter "name=tk_was" | grep -q . && docker stop tk_was && docker rm tk_was || true'
23         }
24       }
25     }
26     // 컨테이너 실행
27     stage('Docker Build and Run') {
28       steps {
29         script {
30           // Docker Compose를 사용해 서비스 빌드 및 실행
31           sh 'docker-compose -f docker-compose.yaml up --build -d'
32         }
33       }
34     }
35   }
36 }

```

## - Gitlab push 감지

```
MINGW64:/c/Users/MZC/Desktop/tcdocker/was

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (main)
$ git init
Initialized empty Git repository in C:/Users/MZC/Desktop/tcdocker/was/.git/

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (master)
$ git remote add origin http://192.168.0.7/kimgh5303/was_cicd.git

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (master)
$ git branch -m master main

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (main)
$ git pull origin main
remote: Enumerating objects: 3, done.
remote: Counting objects: 100% (3/3), done.
remote: Compressing objects: 100% (2/2), done.
remote: Total 3 (delta 0), reused 0 (delta 0), pack-reused 0
Unpacking objects: 100% (3/3), 2.78 KiB | 407.00 KiB/s, done.
From http://192.168.0.7/kimgh5303/was_cicd
 * branch          main      -> FETCH_HEAD
 * [new branch]    main      -> origin/main


MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (main)
$ git add .
warning: in the working copy of '.idea/workspace.xml', LF will be replaced by CRLF the next time Git touches it

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (main)
$ git commit -m "pipeline"
[main b8594a1] pipeline
1 file changed, 96 insertions(+)
create mode 100644 .idea/workspace.xml

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (main)
$ git push origin main
Enumerating objects: 5, done.
Counting objects: 100% (5/5), done.
Delta compression using up to 8 threads
Compressing objects: 100% (3/3), done.
Writing objects: 100% (4/4), 1.74 KiB | 1.74 MiB/s, done.
Total 4 (delta 0), reused 0 (delta 0), pack-reused 0
To http://192.168.0.7/kimgh5303/was_cicd.git
 5e740f5..b8594a1  main -> main

MZC@DESKTOP-VKCL4BI MINGW64 ~/Desktop/tcdocker/was (main)
$ |
```

kimgh5303 / was

 **pipeline**  
kimgh5303 authored just now

b022c6ae

main was +

History Find file Edit Code

README

Auto DevOps enabled

Add LICENSE

Add CHANGELOG

Add CONTRIBUTING

Add Kubernetes cluster

Add Wiki

Configure Integrations

Name	Last commit	Last update
gradle/wrapper	pipeline	just now
src	pipeline	just now
.gitignore	pipeline	just now
Jenkinsfile	pipeline	just now
README.md	Initial commit	3 minutes ago
build.gradle	pipeline	just now
docker-compose.yaml	pipeline	just now
dockerfile	pipeline	just now
gradlew	pipeline	just now

- Webhook 을 통해 push events 전송
- 블루오션을 통해 확인

kimgh-test < 38
파이프라인
변경사항
테스트
아티팩트
로그아웃

브랜치: - 1h 17m 31s 변경사항 없음  
커밋: - - Started by GitLab push by kimgh5303

설명 Started by GitLab push by kimgh5303



Build - 19s

✓	> Check out from version control	2s
✓	> chmod +x ./gradlew — Shell Script	<1s
○	> ./gradlew build -x test — Shell Script	1h 17m 41s

kimgh-test < 38
파이프라인
변경사항
테스트
아티팩트
로그아웃

브랜치: - 1h 18m 47s 변경사항 없음  
커밋: - - Started by GitLab push by kimgh5303

설명 Started by GitLab push by kimgh5303



Prepare Docker Environment - <1s

✓	> docker ps -q --filter "name=tk_was"   grep -q . && docker stop tk_was && docker rm tk_was    true — Shell Script	<1s
---	--	-----

```

root@ubuntu-bionic:~# docker ps
CONTAINER ID   IMAGE                                COMMAND                  CREATED        STATUS        PORTS
b1d67f844699   kimgh5303/tk_was:tier              "java -jar /app.jar"    4 seconds ago Up 2 seconds  0.0.0.0:8080->8080/tcp,
:::8080->8080/tcp
  
```

kimgh-test < 39

파이프라인
변경사항
테스트
아티팩트
로그아웃

브랜치: -
1m 17s

변경사항 없음

커밋: -
-

Started by GitLab push by kimgh5303

설명 Started by GitLab push by kimgh5303



Docker Build and Run - <1s



> docker-compose -f docker-compose.yaml up --build -d -- Shell Script 3s

kimgh-test < 40

파이프라인
변경사항
테스트
아티팩트
로그아웃

브랜치: -
37s

kimgeonho의 변경사항

커밋: -
in a few seconds

Started by GitLab push by kimgh5303

설명 Started by GitLab push by kimgh5303



Docker Build and Run - 9s

[Docker Build and Run 재시작](#)

> docker-compose -f docker-compose.yaml up --build -d -- Shell Script 9s

Was 연결됨

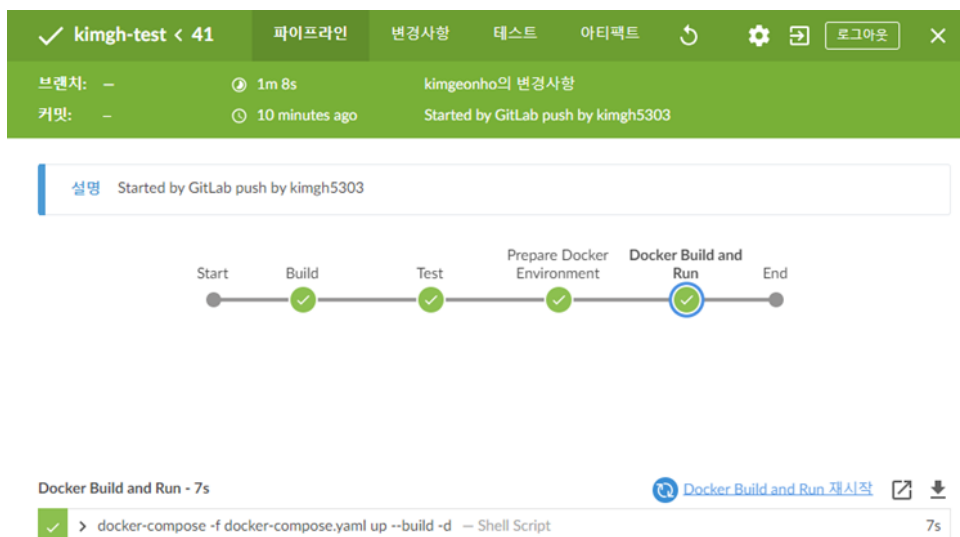


한 번 더 확인해봄

Sysout 코드 작성

```
public class EventService {  
    private final EventRepository eventRepository;  
  
    public ResponseEntity<AllResDto> fetchEvents() {  
        List<Event> results = eventRepository.findAll();  
        System.out.println("pipeline-test2");  
        return new ResponseEntity<>(new AllResDto(true, "f  
    }  
}
```

빌드 및 배포

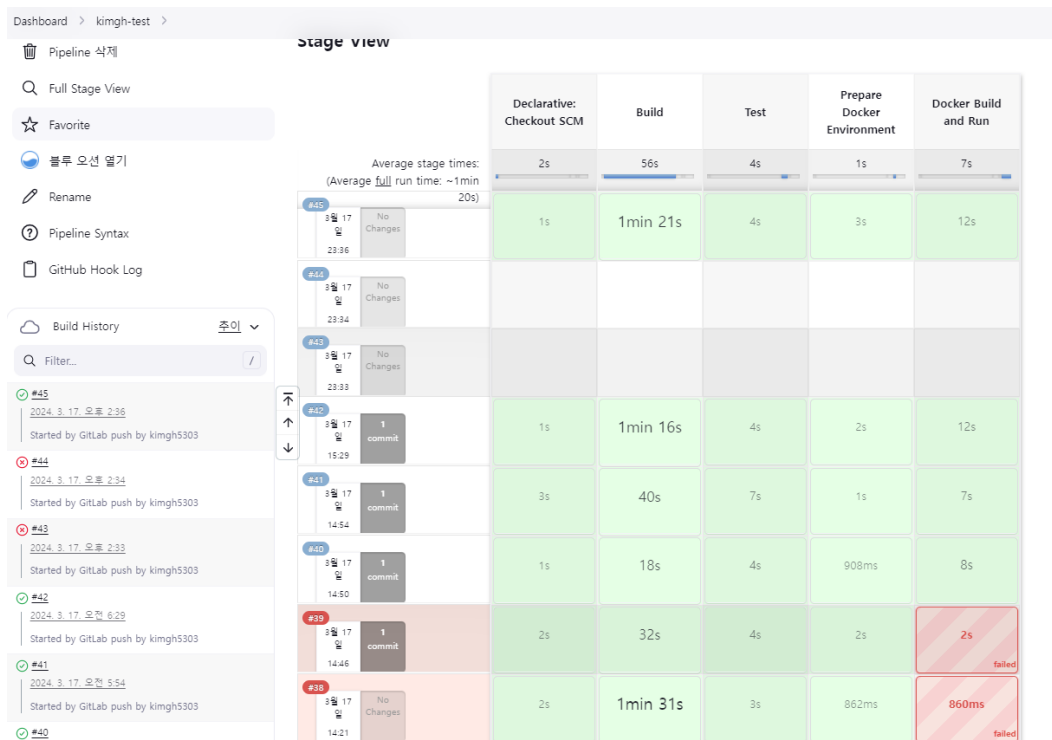




## 로그 확인

```
and u1_0.us_pw=?
Hibernate:
select
    e1_0.et_id,
    e1_0.et_etime,
    e1_0.et_name,
    e1_0.et_stime
from
    events e1_0
pipeline-test2
Hibernate:
select
    e1_0.et_id,
    e1_0.et_etime,
    e1_0.et_name,
    e1_0.et_stime
from
    events e1_0
where
    e1_0.et_id=?
Hibernate:
insert
```

## 대시보드에서 확인



## ● NFS 서버 구축

### # NFS 서버 설정

```
apt-get update
```

### # NFS 서버 설치 및 구성

```
apt-get install -y nfs-kernel-server
```

### # NFS 공유 디렉터리 설정

```
chown nobody:nogroup /home/kgh/Desktop/shared/was
```

```
chmod -R 777 /home/kgh/Desktop/shared
```

```
nano /etc/exports
```

```
/home/kgh/Desktop/shared
```

```
192.168.30.10(rw,sync,no_root_squash,no_subtree_check)
```

```
mount -a
```

### # NFS 서버 설정 적용

```
exportfs -ra
```

```
systemctl restart nfs-kernel-server
```

### # NFS 클라이언트 설정

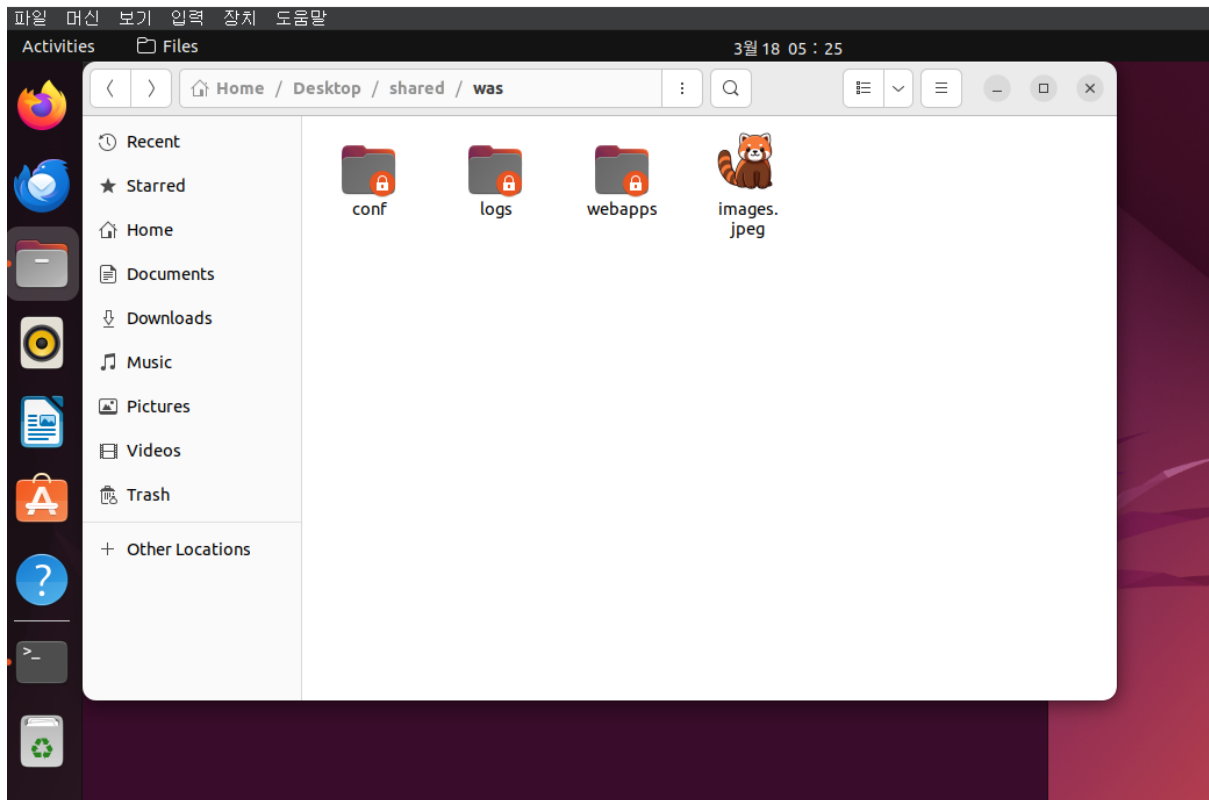
```
apt install -y nfs-common
```

```
mount 192.168.30.20:/home/kgh/Desktop/shared/was /root/was
```

```
nano /etc/fstab
```

```
192.168.30.20:/home/kgh/Desktop/shared/was /root/was nfs defaults 0 0
```

nfs\_server [실행 중] - Oracle VM VirtualBox



```
root@ubuntu-bionic:~/was# ls
conf  images.jpeg  logs  webapps
root@ubuntu-bionic:~/was#
```

## V. 출처

---

<https://gksdudrb922.tistory.com/236>

<https://smattme.com/posts/auto-deploy-spring-boot-app-using-gitlab-ci-cd/>

[https://velog.io/@arin/Gitlab-CICD-](https://velog.io/@arin/Gitlab-CICD-Docker%EB%A1%9C-%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0)

[Docker%EB%A1%9C-%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0](https://velog.io/@arin/Gitlab-CICD-Docker%EB%A1%9C-%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0)

[https://twofootdog.github.io/Docker-](https://twofootdog.github.io/Docker-Docker%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%A1%9C-jenkins-%EC%8B%A4%ED%96%89-%ED%9B%84-%EC%86%8C%EC%8A%A4-push-%EC%8B%9C-%EC%9E%90%EB%8F%99-%EB%B9%8C%EB%93%9C%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0%28Springboot%29/)

[Docker%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%A1%9C-](https://twofootdog.github.io/Docker-Docker%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%A1%9C-jenkins-%EC%8B%A4%ED%96%89-%ED%9B%84-%EC%86%8C%EC%8A%A4-push-%EC%8B%9C-%EC%9E%90%EB%8F%99-%EB%B9%8C%EB%93%9C%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0%28Springboot%29/)

[jenkins-%EC%8B%A4%ED%96%89-%ED%9B%84-%EC%86%8C%EC%8A%A4-](https://twofootdog.github.io/Docker-Docker%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%A1%9C-jenkins-%EC%8B%A4%ED%96%89-%ED%9B%84-%EC%86%8C%EC%8A%A4-push-%EC%8B%9C-%EC%9E%90%EB%8F%99-%EB%B9%8C%EB%93%9C%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0%28Springboot%29/)

[push-%EC%8B%9C-%EC%9E%90%EB%8F%99-%EB%B9%8C%EB%93%9C%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0%28Springboot%29/](https://twofootdog.github.io/Docker-Docker%EC%BB%A8%ED%85%8C%EC%9D%B4%EB%84%88%EB%A1%9C-jenkins-%EC%8B%A4%ED%96%89-%ED%9B%84-%EC%86%8C%EC%8A%A4-push-%EC%8B%9C-%EC%9E%90%EB%8F%99-%EB%B9%8C%EB%93%9C%EB%B0%B0%ED%8F%AC%ED%95%98%EA%B8%B0%28Springboot%29/)

<https://frtt0608.tistory.com/143>

[https://velog.io/@dgh03207/Jenkins-Docker-gitLab-](https://velog.io/@dgh03207/Jenkins-Docker-gitLab-Webhook%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%9E%90%EB%8F%99-%EB%B0%B0%ED%8F%AC)

[Webhook%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%9E%90%EB%8F%99-%EB%B0%B0%ED%8F%AC](https://velog.io/@dgh03207/Jenkins-Docker-gitLab-Webhook%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%9E%90%EB%8F%99-%EB%B0%B0%ED%8F%AC)

<https://www.nasa1515.com/docker-jenkins-ci-build/>

<https://gksdudrb922.tistory.com/236>

<https://docs.gitlab.com/ee/install/docker.html>

<https://jiseok-woo.tistory.com/23>

<https://anfrhrl5555.tistory.com/137>

<https://twofootdog.tistory.com/13>

<https://velog.io/@dgh03207/Jenkins-Docker-gitLab-Webhook%EC%9D%84-%EC%9D%B4%EC%9A%A9%ED%95%9C-%EC%9E%90%EB%8F%99-%EB%B0%B0%ED%8F%AC>

<https://m.blog.naver.com/wideeyed/222079622746>

<https://kim-dragon.tistory.com/255>