



DOKKI Porting Guide



기술 스택

사용한 제품 버전

배포 과정

서버 환경 구성

Docker 설치

보안 및 기타 설정

DNS 설정

SSL 인증

방화벽 설정

Nginx 설치 및 설정

배포 환경 구성

Jenkins 빌드 및 컨테이너 실행

Jenkins 계정 생성 및 플러그인 설치

Jenkins Multibranch Pipeline 프로젝트 추가

Jenkins에 Gitlab 연동

Jenkins에 Docker hub Credential 정보 저장

Jenkins Master/Slave 배포 구조 구성

Redis DB 추가

Mysql DB 추가

환경 변수

외부 서비스 활용

OpenAI 가입 및 활용 방법

알라딘 API 활용 방법

Kakao Login API 활용 방법

기술 스택

사용한 제품 버전

| Cloud

- Amazon AWS EC2
- Google Cloud Platform

IDE

- `IntelliJ IDEA` 2022.3.1
- `VSCode` 1.74.2

DB

- `MySQL` 5.7
- `Redis` 5.0.3

Backend

- `JDK` 11
- `Spring Boot` 2.7.11
- `Spring Data JPA` 2.7.11
- `Spring Data Redis` 2.3.1
- `Spring Cloud Open Feign` 3.1.1
- `Spring Cloud Eureka` 3.1.1
- `Spring Cloud Gateway` 3.1.6
- `Lombok` 1.18.26
- `Swagger UI` 3.0.0
- `Gradle` 7.5.1

FrontEnd

- `Flutter` 3.10.0-12.0
- `Dart` 3.1.0

Deploy & CI/CD

- `Jenkins`

- `Docker` 24.0.0
- `Docker hub`
- `Docker-compose` 1.25.0
- `Nginx` 1.18.0
- `Ubuntu` 20.04

| 소통

- `Discord`
- `Notion`
- `MatterMost`

| 형상 관리

- `Gitlab`
- `Git Bash`
- `SourceTree`

| 이슈 관리

- `Jira`

| 기획

- `Figma`
- `ERD Cloud`

| etc

- `MobaXTerm`

배포 과정

서버 환경 구성

Docker 설치

1. EC2에 사전 패키지 설치를 위한 명령어를 입력한다.

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

2. 도커 설치를 위해 linux 용 gpg key를 다운로드 받는다.

```
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
```

3. Docker를 설치한다.

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

보안 및 기타 설정

DNS 설정

1. 가비아에서 DNS를 발급 신청

전체 1건

≡ 10 20 50 100

☐ 도메인

dokki.kr

2023-04-21 ~ 2024-04-21 (D-361) [연장 >](#)

20,000원/년


[관리](#)

2. 클릭해서 세부정보를 조회 → DNS 연결 > 설정

DNS 정보		
도메인 연결		설정
포워딩		설정
파킹	웹	설정
	모바일	설정
DNS 레코드 설정		설정

3. 레코드 수정을 클릭

dokki.kr

• 레코드 개수 : 1개 • 최근 업데이트 : 2023-04-26 14:59:42 • 네임서버 :  ns.gabia.co.kr

DNS 설정

레코드 수정

4. cmd 창에서 도메인에 대한 IP 주소를 조회 : `nslookup {도메인}`

```
C:\Users\SSAFY>nslookup k8e204.p.ssafy.io
Server:   kns.kornet.net
Address:  168.126.63.1

Non-authoritative answer:
Name:     k8e204.p.ssafy.io
Address:  13.125.238.197
```

5. 아래와 같이 설정 > 저장 클릭.

DNS 레코드 수정

타입	호스트	값/위치	TTL	우선 순위	서비스	상태
A	@	13.125.238.197	600		DNS 설정	<div>수정</div> <div>삭제</div>

+ 레코드 추가

저장

취소

SSL 인증

1. ubuntu 18.04 이상 버전에서는 snap 이 설치되어 있음. 없으면 설치.
2. 아래 명령어 차례로 입력
 - 순서대로
 1. 개인 이메일 작성
 2. 약관 동의 → Y
 3. 소식 이메일로 수신 동의 여부 → N
 4. 도메인 이름 → 예) `k8e2041.p.ssafy.io`

```

sudo snap install core; sudo snap refresh core

sudo snap install --classic certbot

sudo ln -s /snap/bin/certbot /usr/bin/certbot

# Stop your webserver, then run this command to get a certificate.
sudo certbot certonly --standalone

```

3. 키 생성 문구 확인

```

Successfully received certificate.
Certificate is saved at: /etc/letsencrypt/live/k8e2041.p.ssafy.io/fullchain.pem
Key is saved at: /etc/letsencrypt/live/k8e2041.p.ssafy.io/privkey.pem
This certificate expires on 2023-08-16.
These files will be updated when the certificate renews.
Certbot has set up a scheduled task to automatically renew this certificate in the background.

-----
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
-----

```

방화벽 설정

1. ufw 설치 및 default로 들어오는 패킷을 차단 설정

```

sudo apt install ufw
sudo ufw default deny

```

2. ssh, http, jenkins, https 포트 허용 및 ufw 방화벽 활성화

```

sudo ufw allow 22
sudo ufw allow 80
sudo ufw allow 8080
sudo ufw allow 443
sudo ufw enable

```

3. 현재 방화벽 상태 조회

```

sudo ufw status

```

```

Status: active

```

To	Action	From
--	-----	----
22	ALLOW	Anywhere
80	ALLOW	Anywhere
8080	ALLOW	Anywhere
443	ALLOW	Anywhere
22 (v6)	ALLOW	Anywhere (v6)
80 (v6)	ALLOW	Anywhere (v6)
8080 (v6)	ALLOW	Anywhere (v6)
443 (v6)	ALLOW	Anywhere (v6)

Nginx 설치 및 설정

1. Nginx 설치 및 실행

```
# 서버의 패키지 목록 업데이트
sudo apt update
sudo apt upgrade
sudo apt autoremove

# Nginx 설치
sudo apt install nginx

# Nginx 상태 확인
sudo service nginx start && \
sudo service nginx status

# Nginx Version 확인
sudo dpkg -l nginx
nginx -v
```

2. ssl 인증 설정 추가

```
# default nginx 설정 수정 - ssl 인증서 연결
sudo vi /etc/nginx/sites-available/default
```

3. 아래처럼 리버스 프록시 설정

```
server {
    listen 80    default_server;
    listen [::]:80    default_server;

    root /var/www/example.com;
    index index.html;

    location / {
        try_files $uri $uri/ =404;
    }
}

server {
    listen 443    ssl;
    listen [::]:443    ssl;
    server_name    dokki.kr;
```



```

ssl      on;
ssl_certificate      /etc/letsencrypt/live/dokki.kr/fullchain.pem;
ssl_certificate_key  /etc/letsencrypt/live/dokki.kr/privkey.pem;

root /var/www/html;

# Add index.php to the list if you are using PHP
index index.html index.htm index.nginx-debian.html;

server_name _;

location / {
    proxy_pass http://gateway/;
    proxy_redirect      off;
    proxy_set_header    Host $host;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    add_header 'Access-Control-Allow-Origin' '*';
}
location /api/ {
    proxy_pass http://localhost:8080/;
    proxy_redirect      off;
    proxy_set_header    Host $host;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    add_header 'Access-Control-Allow-Origin' '*';
}
location /eureka {
    proxy_pass http://localhost:8761/;
    proxy_redirect      off;
    proxy_set_header    Host $host;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    add_header 'Access-Control-Allow-Origin' '*';
}
location /swagger/ {
    proxy_pass http://localhost:5000/;
    proxy_redirect      off;
    proxy_set_header    Host $host;
    proxy_set_header    X-Real-IP $remote_addr;
    proxy_set_header    X-Forwarded-For $proxy_add_x_forwarded_for;
    add_header 'Access-Control-Allow-Origin' '*';
    add_header 'Access-Control-Allow' '*';
}
location /images {
    alias /home/ubuntu/resources/images;
}
}

```

4. `sudo service nginx reload` 로 nginx 를 reload해서 변경 사항 적용

배포 환경 구성

Jenkins 빌드 및 컨테이너 실행

- 서버 스펙 요구 조건 확인

Prerequisites

Minimum hardware requirements:

- 256 MB of RAM
- 1 GB of drive space (although 10 GB is a recommended minimum if running Jenkins as a Docker container)

Recommended hardware configuration for a small team:

- 4 GB+ of RAM
- 50 GB+ of drive space

1. 인스턴스에 작업 디렉토리 생성

```
mkdir dokki
cd dokki
```

2. Jenkins 이미지를 만들기 위해 `jenkins-docker-compose.yml` 파일을 vim으로 열고, 아래 내용을 입력한 후 `:wq` 로 저장

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

```
version: '3.5'

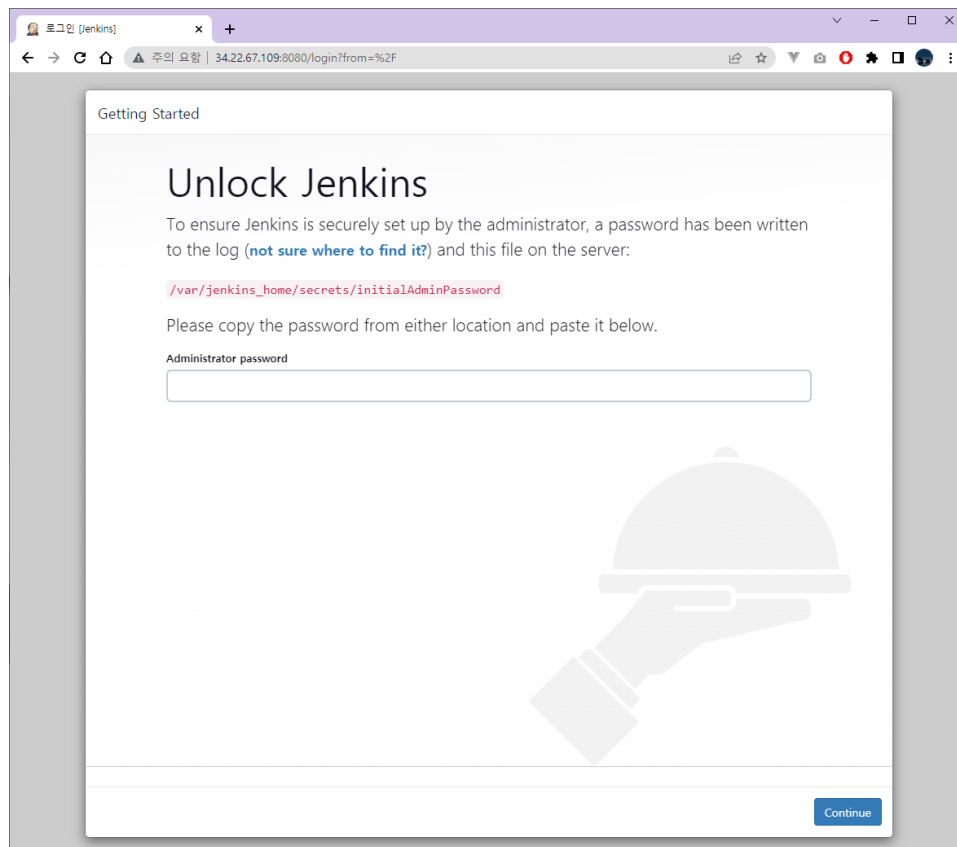
services:
  # 서비스 명
  jenkins:
    image: jenkins/jenkins:lts
    container_name: jenkins
    build: .
    environment:
      - JENKINS_OPTS=--httpPort=8080
    volumes:
      - /var/run/docker.sock:/var/run/docker.sock # DooD 방식, 도커 소켓 공유
      - /home/ubuntu/dokki/jenkins:/var/jenkins_home # jenkins 설치 폴더
```

```

- /home/ubuntu/dev/config:/var/conf
- /home/ubuntu/dokki/app:/app
environment:
- TZ=Asia/Seoul # 타임존 설정
ports:
- "8080:8080" # jenkins 접속 port
privileged: true
user: root

```

3. `sudo docker-compose -f jenkins-docker-compose.yml up -d` 로 실행
4. GCP에서 방화벽 규칙 설정까지 완료하고 난 후, `http://{domain or public ip}:8080` 접속 시도



Jenkins 계정 생성 및 플러그인 설치

1. `sudo docker logs jenkins` 로 비밀번호를 확인

```

*****
*****
*****

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

517411e12d65430ca840a89de942792a

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

*****
*****
*****

```

2. 비밀번호 입력 > 로그인

Getting Started

Unlock Jenkins

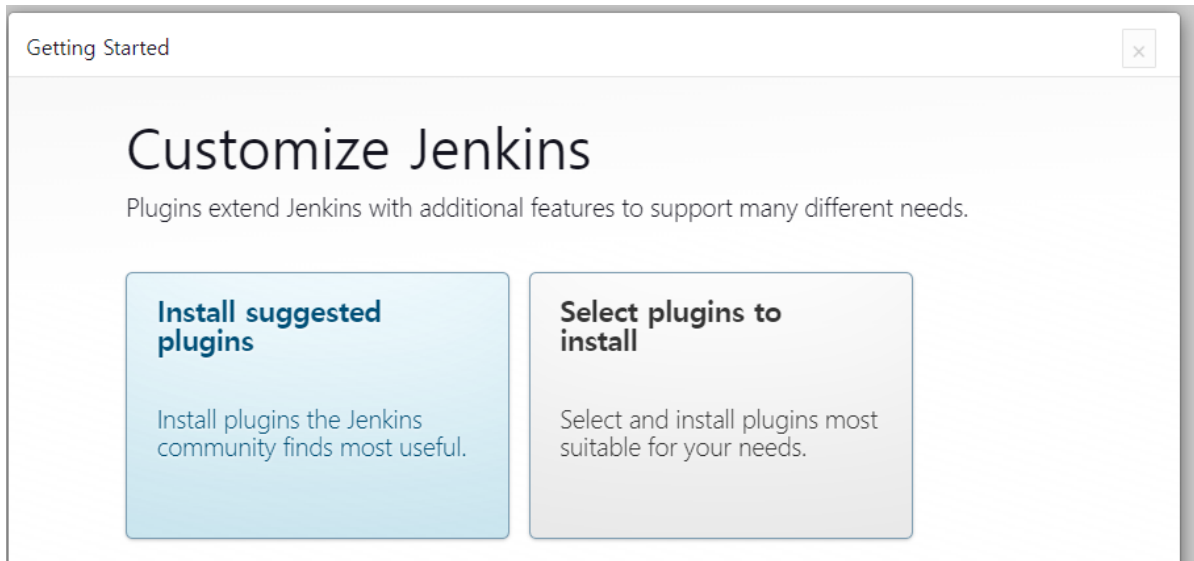
To ensure Jenkins is securely set up by the administrator, a password has been written to the log (**not sure where to find it?**) and this file on the server:

`/var/jenkins_home/secrets/initialAdminPassword`

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

Administrator password

3. `Install Suggested plugins` 를 선택



✓ Folders	✓ OWASP Markup Formatter	✓ Build Timeout	✓ Credentials Binding
✓ Timestampers	✓ Workspace Cleanup	✓ Ant	✓ Gradle
⌚ Pipeline	⌚ GitHub Branch Source	⌚ Pipeline: GitHub Groovy Libraries	⌚ Pipeline: Stage View
⌚ Git	⌚ SSH Build Agents	○ Matrix Authorization Strategy	⌚ PAM Authentication
⌚ LDAP	⌚ Email Extension	✓ Mailer	

이런 플러그인들이 설치됨.

4. 플러그인들이 설치되고 나면, 관리자 계정을 만들고 `save and continue` > `save and finish` > `start using jenkins` 를 클릭

Getting Started

Create First Admin User

계정명

dokki

암호

.....

암호 확인

.....

이름

jcw

이메일 주소

codnjs8221@naver.com

Jenkins 2.387.2

Skip and continue as admin

Save and Continue

5. Jenkins 관리 > 플러그인 관리를 클릭

새로운 Item

사람

빌드 기록

Jenkins 관리

My Views

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중

2 대기 중

Jenkins 관리

Building on the built-in node can be a security issue. You should set up distributed builds. See [the documentation](#).

Set up agent

Set up cloud

Dismiss

System Configuration

시스템 설정

환경 변수 및 경로 정보등을 설정합니다.

Global Tool Configuration

Configure tools, their locations and automatic installers.

플러그인 관리

Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.

노드 관리

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

6. Gitlab 관련 플러그인을 설치. > **Install without restart** 를 클릭

Updates

Available plugins

Installed plugins

Advanced settings

Download progress

Plugins

gitlab

1

Install	Name ↓	Released
<input checked="" type="checkbox"/>	<div>GitLab 1.6.0</div> <div>Build Triggers</div> <div>This plugin allows GitLab to trigger Jenkins builds and display their results in the GitLab UI.</div> <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div>	2 mo 15 days ago
<input checked="" type="checkbox"/>	<div>Generic Webhook Trigger 1.86.2</div> <div>notification github webhook Build Parameters gitlab Build Triggers bitbucket bitbucket-server jira</div> <div>Can receive any HTTP request, extract any values from JSON or XML and trigger a job with those values available as variables. Works with GitHub, GitLab, Bitbucket, Jira and many more.</div>	21 days ago
<input checked="" type="checkbox"/>	<div>GitLab API 5.0.1-78.v47a_45b_9f78b_7</div> <div>Library plugins (for use by other plugins)</div> <div>This plugin provides GitLab API for other plugins.</div>	6 mo 5 days ago
<input checked="" type="checkbox"/>	<div>GitLab Authentication 1.16</div> <div>Authentication and User Management</div> <div>This is the an authentication plugin using gitlab OAuth.</div> <div>This plugin is up for adoption! We are looking for new maintainers. Visit</div>	9 mo 12 days ago

Install without restart

Download now and install after restart

Update information obtained: 20 min ago

7. 마찬가지로 Docker 관련 플러그인을 설치

Plugins

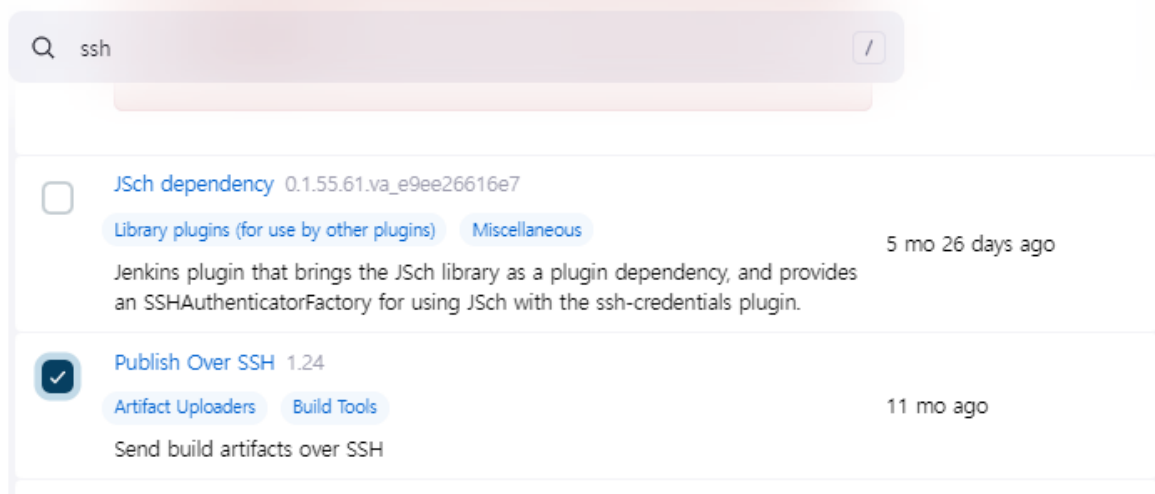
Install	Name ↓	Released
<input checked="" type="checkbox"/>	Docker 1.3.0 Cloud Providers Cluster Management docker This plugin integrates Jenkins with Docker <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div>	1 day 14 hr ago
<input checked="" type="checkbox"/>	Docker Commons 1.21 Library plugins (for use by other plugins) docker Provides the common shared functionality for various Docker-related plugins.	5 mo 3 days ago
<input checked="" type="checkbox"/>	Docker Pipeline 563.vd5d2e5c4007f pipeline DevOps Deployment docker Build and use Docker containers from pipelines. <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div>	1 mo 25 days ago
<input checked="" type="checkbox"/>	Docker API 3.2.13-37.vf3411c9828b9 Library plugins (for use by other plugins) docker This plugin provides docker-java API for other plugins. <div>This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.</div>	9 mo 12 days ago

Install without restart

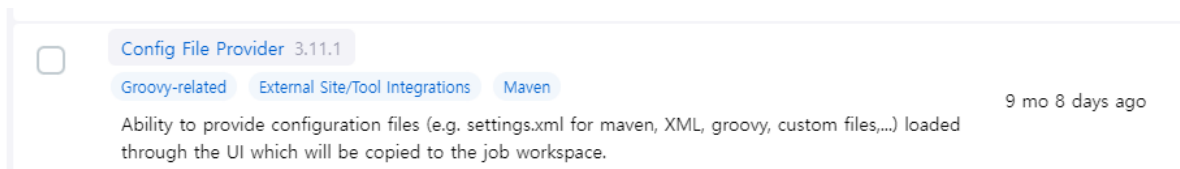
Download now and install after restart

Update information obtained: 21 min ago

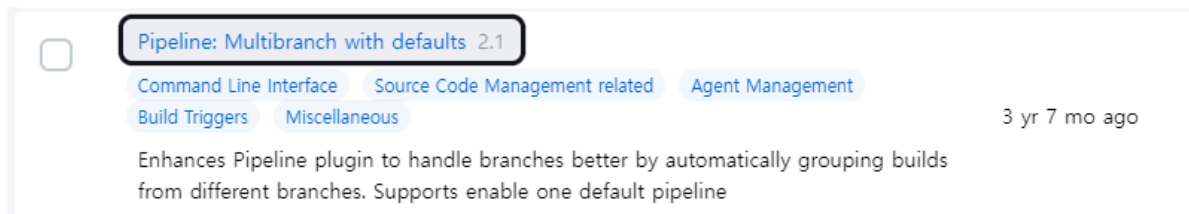
8. 마찬가지로 SSH 관련 플러그인을 설치



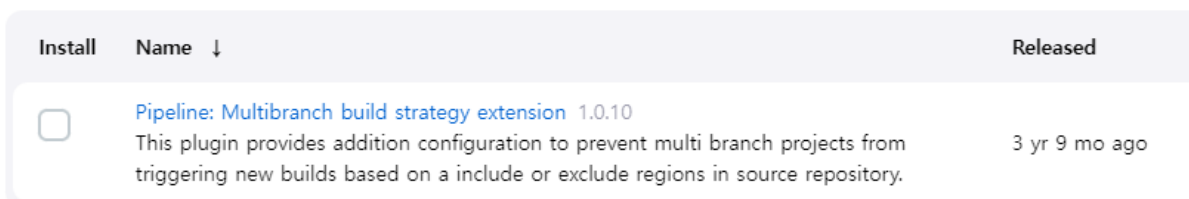
9. **Config File Provider** 플러그인 설치: “Dashboard > Jenkins 관리”에서 **Managed Files** 항목이 생김. 내장 프로젝트에서 JenkinsFile을 관리할 수 있음.

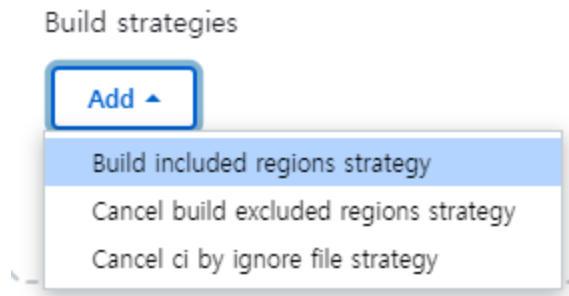


10. **Pipeline: Multibranch with defaults** 플러그인 설치 : Multibranch project에서 build Configuration > Add에 **with default Jenkinsfile** 옵션이 추가



11. **Pipeline: Multibranch build strategy extension** 플러그인 설치 : Multibranch project에서 build strategies > Add에 옵션이 추가





12. `SSH Build Agents` 플러그인 설치
13. `Multibranch Scan Webhook Trigger` 플러그인 설치

Jenkins Multibranch Pipeline 프로젝트 추가

1. New Item > multibranch pipeline 프로젝트 생성

Enter an item name

dokki

» A job already exists with the name 'dokki'

**Freestyle project**

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

**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 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.

**Folder**

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

**Multibranch Pipeline with defaults**

Extend Multibranch pipeline plugin and build branches it the default preped pipeline script. Creates a set of Pipeline projects according to detected branches in one SCM repository.

2. Add Source > Single repository & branch을 클릭

Add source ▲

Git

GitHub

Single repository & branch

3. Branch Sources > Credentilals > Jenkins 클릭해서 Credential을 등록

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted) ▼

Kind

Username with password ▼

Scope ?

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

Username ?

jcw

☐ Treat username as secret ?

Password ?

.....

ID ?

4. 아래 처럼 항목을 채우고, Name 항목과 Branch 항목을 Gitlab에 생성한 서비스 별 각 브랜치마다 따로 작성하여 똑같이 작성.

본 프로젝트에서는 BE-USER, BE-TIMER, BE-BOOK, BE-REVIEW, BE-GATEWAY, BE-EUREKA 로 총 6개를 추가하였음.

Single repository & branch

Allows a single, fixed branch of some repository to be configured. Not normally used, as it does not allow branches to be detected automatically. Might be used, for example, if you want to build all branches of one repository, *and* the master branch of a special clone.

Name ?

BE-USER

Source Code Management

Git

Repositories ?

Repository URL ?

https://lab.ssafy.com/s08-final/S08P31E204.git

Credentials ?

jcw/*****

Add

고급

Add Repository

Branches to build ?

Branch Specifier (blank for 'any') ?

*/BE-USER

Add Branch

5. Build Configuration에서 default JenkinsFile을 “Dokki-Multibranch-Jenkinsfile”라고 등록. 여기서 등록하는 JenkinsFile은 이후 과정에서 추가할 예정임.

Build Configuration

Mode

by default Jenkinsfile

Script ID ?

Dokki-Multibranch-Jenkinsfile

☒ Run default Jenkinsfile within Groovy sandbox ?

6. 설정을 저장하고 나가면 이런식으로 브랜치 별 파이프라인이 생성됨

✓	☀	BE-BOOK	3 hr 2 min #58	12 days #29	30 sec	▶	★
⊗	☀	BE-EUREKA	—	—	—	▶	★
✓	☀	BE-GATEWAY	3 hr 3 min #18	9 days 20 hr #2	22 sec	▶	★
✓	☀	BE-REVIEW	3 hr 2 min #73	6 days 15 hr #49	3 min 7 sec	▶	★
✓	☀	BE-TIMER	3 hr 2 min #35	1 day 16 hr #30	3 min 12 sec	▶	★
✓	☀	BE-USER	3 hr 2 min #68	2 days 19 hr #59	3 min 13 sec	▶	★

7. Dashboard > Manage Jenkins > Managed Files 클릭

System Configuration



시스템 설정

환경변수 및 경로 정보등을 설정합니다.



Global Tool Configuration

Configure tools, their locations and automatic installers.



플러그인 관리

Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.



노드 관리

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.



Managed files

e.g. settings.xml for maven, central managed scripts, custom files, ...

8. Add a new Config > Groovy file 클릭, 아까 작성한 “Dokki-Multibranch-Jenkinsfile” ID 넣고 Next

Manage Jenkins

→ Config Files

+ Add a new Config

New configuration

Type

- ☐ Global Maven settings.xml
A global maven settings.xml which can be referenced within Apache Maven jobs.
Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)
- ☐ Maven settings.xml
A settings.xml which can be referenced within Apache Maven jobs.
Use it within maven projects or maven builder and reference credentials for a server authentication from here: [credentials](#)
- ☐ Properties file
a Properties file [credentials](#)
- ☐ Json file
a Json file
- ☐ Maven toolchains.xml
a toolchains.xml which can be referenced within Apache Maven jobs
- ☐ Simple XML file
a general xml file
- ☒ Groovy file
a reusable groovy script
- ☐ Custom file
a custom file (e.g. text or any other not yet available format)
- ☐ Extended Email Publisher Groovy Template
A Groovy template used by the Extended Email Publisher plugin to generate emails.
- ☐ Extended Email Publisher Jelly Template
A Jelly template used by the Extended Email Publisher plugin to generate emails.

ID

ID of the config file

Dokki-Multibranch-Jenkinsfile

9. Content에 다음과 같이 작성

비밀번호와 API Token 정보는 제외했다.

```
pipeline{
  environment{
    DOCKER_HUB_REPOSITORY = 'noonmap/dokki_repo'
    DOCKER_HUB_CREDENTIALS = credentials('docker-hub-noonmap')
  }

  agent {
    label 'master'
  }
}
```

```

stages{
  stage('Fetch'){
    steps{
      sh "echo 'Fetch From gitlab'"
      sh "echo 'branch: ${BRANCH_NAME}'"
      git branch: env.BRANCH_NAME, credentialsId: 'dokki', url: 'https://lab.ssafy.
com/s08-final/S08P31E204.git'
    }
  }
  stage('Docker Hub Login'){
    steps{
      sh "echo $DOCKER_HUB_CREDENTIALS_PSW | docker login -u $DOCKER_HUB_CREDENTI
ALS_USR --password-stdin"
    }
  }
  // 브랜치 별로 실행하기
  stage('Build'){
    parallel{ // parallel하게 진행 (BE, develop의 경우)
      stage('Build-Gateway'){
        when{
          anyOf{
            branch 'BE-GATEWAY';
            branch 'develop'
          }
        }
        steps{
          sh 'docker build --build-arg SERVICE_NAME=gateway --tag ${DOCKER_HUB_
REPOSITORY}:gateway ./backend'
          sh 'echo "Run Gateway"'

          sh 'docker push ${DOCKER_HUB_REPOSITORY}:gateway'
        }
      }
      stage('Build-Eureka'){
        when{
          anyOf{
            branch 'BE-EUREKA';
            branch 'develop'
          }
        }
        steps{
          // eureka build & deploy
          sh 'docker build --tag ${DOCKER_HUB_REPOSITORY}:eureka ./backend/eure
ka'
          sh 'echo "Run Eureka"'

          sh 'docker push ${DOCKER_HUB_REPOSITORY}:eureka'
        }
      }
      stage('Build-Timer'){
        when{
          anyOf{
            branch 'BE-TIMER';

```



```

        branch 'develop'
    }
}
steps{
    sh 'docker build --build-arg SERVICE_NAME=timer --tag ${DOCKER_HUB_RE
    POSITORY}:timer ./backend'
    sh 'echo "Run Timer-Service"'

    sh 'docker push ${DOCKER_HUB_REPOSITORY}:timer'
}
}
stage('Build-User'){
    when{
        anyOf{
            branch 'BE-USER';
            branch 'develop'
        }
    }
    steps{
        sh 'docker build --build-arg SERVICE_NAME=user --tag ${DOCKER_HUB_REP
        OSITORY}:user ./backend'
        sh 'echo "Run User-Service"'

        sh 'docker push ${DOCKER_HUB_REPOSITORY}:user'
    }
}
stage('Build-Book'){
    when{
        anyOf{
            branch 'BE-BOOK';
            branch 'develop'
        }
    }
    steps{
        sh 'docker build --build-arg SERVICE_NAME=book --tag ${DOCKER_HUB_REP
        OSITORY}:book ./backend'
        sh 'echo "Run Book-Service"'

        sh 'docker push ${DOCKER_HUB_REPOSITORY}:book'
    }
}
stage('Build-Review'){
    when{
        anyOf{
            branch 'BE-REVIEW';
            branch 'develop'
        }
    }
    steps{
        sh 'docker build --build-arg SERVICE_NAME=review --tag ${DOCKER_HUB_R
        EPOSITORY}:review ./backend'
        sh 'echo "Run Review-Service"'

        sh 'docker push ${DOCKER_HUB_REPOSITORY}:review'
    }
}

```

```

    }
  }
}
stage('Create Docker Network'){
  agent {
    label 'agent1'
  }
  steps{
    sh '''
      if(docker network ls | grep "dokki-network"); then echo "ok"; else docker n
etwork create dokki-network; fi
    '''
  }
}
// 변수를 쓰려면 script{}로 감싸줘야 함
// 변수를 sh안에서 쓰려면 반드시 작은 따옴표가 아니라 큰 따옴표로 감싸야 한다. '''(x) ""(o)
stage('Execute-Eureka'){
  // eureka는 한번 다시 실행하면 나머지 모든 서비스를 다시 시작.
  agent{
    label 'agent1'
  }
  when{
    anyOf{
      branch 'BE-EUREKA';
      branch 'develop'
    }
  }
  steps {
    script{
      sh 'docker pull noonmap/dokki_repo:eureka'
      sh 'echo "Pull Eureka"'
      // eureka build & deploy
      sh """
        if (docker ps | grep "eureka-service"); then docker stop eureka-service;
        if (docker ps -a | grep "eureka-service"); then docker rm eureka-service;

        docker run -itd -p 8761:8761 \
          --net dokki-network \
          --name eureka-service noonmap/dokki_repo:eureka
          """
      sh 'echo "Run Eureka"'
    }
  }
  post{
    success{
      discordSend description: "${BRANCH_NAME}의 변경사항이 반영되었습니다.",
        footer: "배포가 성공하였습니다.\nCommit 내역 확인하기 : `git show ${GIT_COMMIT}`",
        link: env.BUILD_URL, result: currentBuild.currentResult,
        title: "Eureka 배포",
        webhookURL: "https://discord.com/api/webhooks/1102411269262278738/V3K-UIBok
EuiLGtj003xpuLuzwbRgaeIoDTioJRdG5uPRmtbg_3g0mxvCUYFb0voi0Am"
    }
  }
}

```

```

    }
  }
  stage('Execute-Gateway'){
    agent{
      label 'agent1'
    }
    when{
      anyOf{
        branch 'BE-GATEWAY';
        branch 'BE-EUREKA';
        branch 'develop'
      }
    }
    steps{
      script{
        def SERVICE_NAME="gateway"
        def PORT="8080"
        sh """docker pull noonmap/dokki_repo:${SERVICE_NAME}"""
        sh """echo \"Deploy ${SERVICE_NAME}-Service\""""
        sh """
          if (docker ps | grep "${SERVICE_NAME}-service"); then docker stop ${SERVICE_NAME}-service; fi
          if (docker ps -a | grep "${SERVICE_NAME}-service"); then docker rm ${SERVICE_NAME}-service; fi
          docker run -itd -p ${PORT}:8080 \
            --env "eureka.client.serviceUrl.defaultZone=http://eureka-service:8761/eureka/" \
            --net dokki-network \
            --name ${SERVICE_NAME}-service noonmap/dokki_repo:${SERVICE_NAME}
          """
        sh 'echo "Run ${SERVICE_NAME}-Service"'
      }
    }
    post{
      success{
        discordSend description: "${BRANCH_NAME}의 변경사항이 반영되었습니다.",
          footer: "배포가 성공하였습니다.\nCommit 내역 확인하기 : `git show ${GIT_COMMIT}`",
          link: env.BUILD_URL, result: currentBuild.currentResult,
          title: "Gateway 배포",
          webhookURL: "https://discord.com/api/webhooks/1102411269262278738/V3K-UIBokEuILGtj003xpuLuzwbRgaeIoDTioJRdG5uPRmtbg_3g0mxvCUYFb0voi0Am"
      }
    }
  }
  stage('Execute-User'){
    agent{
      label 'agent1'
    }
    when{
      anyOf{
        branch 'BE-USER';
        branch 'BE-EUREKA';
        branch 'develop'
      }
    }
  }

```

```

    }
    steps{
        script{
            def SERVICE_NAME="user"
            def PORT="5200"
            sh """docker pull noonmap/dokki_repo:${SERVICE_NAME}"""
            sh """echo \"Deploy ${SERVICE_NAME}-Service\""""
            sh """
                if (docker ps | grep "${SERVICE_NAME}-service"); then docker stop ${SERVI
CE_NAME}-service; fi
                if (docker ps -a | grep "${SERVICE_NAME}-service"); then docker rm ${SERV
ICE_NAME}-service; fi
                docker run -itd -p ${PORT}:8080 \
                    --env DB_URL=jdbc:mysql://${SERVICE_NAME}-mysql/dokki?autoReconnect=true
\
                    --env DB_USER=dokki \
                    --env DB_PASSWD={MySQL 비밀번호 삽입} \
                    --env PORT=${PORT} \
                    --env "eureka.client.serviceUrl.defaultZone=http://eureka-service:8761/eu
reka/" \
                    --env REDIS_HOST=user-redis \
                    --env REDIS_PORT=6379 \
                    --env REDIS_PASSWORD={Redis 비밀번호 삽입} \
                    --env UPLOAD_PATH=/home/resources \
                    --env UPLOAD_DIR=images \
                    --env HOST_URI=https://dokki.kr \
                    --volume /home/ubuntu/resources:/home/resources \
                    --net dokki-network \
                    --name ${SERVICE_NAME}-service noonmap/dokki_repo:${SERVICE_NAME}
                    """
                sh 'echo "Run ${SERVICE_NAME}-Service"'
            }
        }
        post{
            success{
                discordSend description: "${BRANCH_NAME}의 변경사항이 반영되었습니다.",
                footer: "배포가 성공하였습니다.\nCommit 내역 확인하기 : `git show ${GIT_COMMIT}`",
                link: env.BUILD_URL, result: currentBuild.currentResult,
                title: "User-Service 배포",
                webhookURL: "https://discord.com/api/webhooks/1102411269262278738/V3K-UIBok
EuILGtj003xpuLuzwbRgaeIoDTioJRdG5uPRmtbg_3g0mxvCUYFb0voi0Am"
            }
        }
    }
    stage('Execute-Book'){
        agent{
            label 'agent1'
        }
        when{
            anyOf{
                branch 'BE-BOOK';
                branch 'BE-EUREKA';
                branch 'develop'
            }
        }
    }

```

```

    }
    steps{
      script{
        def SERVICE_NAME="book"
        def PORT="5100"
        sh ""docker pull noonmap/dokki_repo:${SERVICE_NAME}""
        sh ""echo \"Deploy ${SERVICE_NAME}-Service\"""
        sh ""
        if (docker ps | grep "${SERVICE_NAME}-service"); then docker stop ${SERVI
CE_NAME}-service; fi
        if (docker ps -a | grep "${SERVICE_NAME}-service"); then docker rm ${SERV
ICE_NAME}-service; fi
        docker run -itd -p ${PORT}:8080 \
          --env DB_URL=jdbc:mysql://${SERVICE_NAME}-mysql/dokki?autoReconnect=true
\
          --env DB_USER=dokki \
          --env DB_PASSWD={MySQL 비밀번호 삽입} \
          --env PORT=${PORT} \
          --env "eureka.client.serviceUrl.defaultZone=http://eureka-service:8761/eu
reka/" \
          --env UPLOAD_PATH=/home/resources \
          --env UPLOAD_DIR=images \
          --env HOST_URI=https://dokki.kr \
          --volume /home/ubuntu/resources:/home/resources \
          --net dokki-network \
          --name ${SERVICE_NAME}-service noonmap/dokki_repo:${SERVICE_NAME}
          ""
        sh 'echo "Run ${SERVICE_NAME}-Service"'
      }
    }
    post{
      success{
        discordSend description: "${BRANCH_NAME}의 변경사항이 반영되었습니다.",
        footer: "배포가 성공하였습니다.\nCommit 내역 확인하기 : `git show ${GIT_COMMIT}`",
        link: env.BUILD_URL, result: currentBuild.currentResult,
        title: "Book-Service 배포",
        webhookURL: "https://discord.com/api/webhooks/1102411269262278738/V3K-UIBok
EuILGtj003xpuLuzwbRgaeIoDTioJRdG5uPRmtbg_3g0mxvCUYFb0voi0Am"
      }
    }
  }
  stage('Execute-Timer'){
    agent{
      label 'agent1'
    }
    when{
      anyOf{
        branch 'BE-TIMER';
        branch 'BE-EUREKA';
        branch 'develop'
      }
    }
  }
  steps{
    script{

```

```

        def SERVICE_NAME = "timer"
        def PORT="5400"
        sh ""docker pull noonmap/dokki_repo:${SERVICE_NAME}""
        sh ""echo \"Deploy ${SERVICE_NAME}-Service\"""
        sh ""
        if (docker ps | grep "${SERVICE_NAME}-service"); then docker stop ${SERVI
CE_NAME}-service; fi
        if (docker ps -a | grep "${SERVICE_NAME}-service"); then docker rm ${SERV
ICE_NAME}-service; fi
        docker run -itd -p ${PORT}:8080 \
        --env DB_URL=jdbc:mysql://${SERVICE_NAME}-mysql/dokki?autoReconnect=true
        \
        --env DB_USER=dokki \
        --env DB_PASSWD={MySQL 비밀번호 삽입} \
        --env PORT=${PORT} \
        --env REDIS_HOST=timer-redis \
        --env REDIS_PORT=6379 \
        --env REDIS_PASSWORD={Redis 비밀번호 삽입} \
        --env "eureka.client.serviceUrl.defaultZone=http://eureka-service:8761/eu
reka/" \
        --net dokki-network \
        --name ${SERVICE_NAME}-service noonmap/dokki_repo:${SERVICE_NAME}
        ""
        sh 'echo "Run ${SERVICE_NAME}-Service"'
    }
}
post{
    success{
        discordSend description: "${BRANCH_NAME}의 변경사항이 반영되었습니다.",
        footer: "배포가 성공하였습니다.\nCommit 내역 확인하기 : `git show ${GIT_COMMIT}`",
        link: env.BUILD_URL, result: currentBuild.currentResult,
        title: "Timer-Service 배포",
        webhookURL: "https://discord.com/api/webhooks/1102411269262278738/V3K-UIBok
EuilGtj003xpuLuzwbRgaeIoDTioJRdG5uPRmtbg_3g0mxvCUYFb0voi0Am"
    }
}
}
stage('Execute-Review'){
    agent{
        label 'agent1'
    }
    when{
        anyOf{
            branch 'BE-REVIEW';
            branch 'BE-EUREKA';
            branch 'develop'
        }
    }
}
steps{
    script{
        def SERVICE_NAME = "review"
        def PORT="5300"
        sh ""docker pull noonmap/dokki_repo:${SERVICE_NAME}""
        sh ""echo \"Deploy ${SERVICE_NAME}-Service\"""
    }
}

```

```

        sh """
        if (docker ps | grep "${SERVICE_NAME}-service"); then docker stop ${SERVICE_NAME}-service; fi
        if (docker ps -a | grep "${SERVICE_NAME}-service"); then docker rm ${SERVICE_NAME}-service; fi
        docker run -itd -p ${PORT}:8080 \
        --env DB_URL=jdbc:mysql://${SERVICE_NAME}-mysql/dokki?autoReconnect=true \
        --env DB_USER=dokki \
        --env DB_PASSWD={MySQL 비밀번호 삽입} \
        --env PORT=${PORT} \
        --env REDIS_HOST=review-redis \
        --env REDIS_PORT=6379 \
        --env REDIS_PASSWORD={Redis 비밀번호 삽입} \
        --env "eureka.client.serviceUrl.defaultZone=http://eureka-service:8761/eureka/" \
        --env UPLOAD_PATH=/home/resources \
        --env UPLOAD_DIR=images \
        --env HOST_URI=https://dokki.kr \
        --env OPENAI_API_TOKEN={ChatGPT Token 삽입} \
        --volume /etc/localtime:/etc/localtime \
        --volume /home/ubuntu/resources:/home/resources \
        --net dokki-network \
        --name ${SERVICE_NAME}-service noonmap/dokki_repo:${SERVICE_NAME}
        """
        sh 'echo "Run ${SERVICE_NAME}-Service"'
    }
}
post{
    success{
        discordSend description: "${BRANCH_NAME}의 변경사항이 반영되었습니다.",
        footer: "배포가 성공하였습니다.\nCommit 내역 확인하기 : `git show ${GIT_COMMIT}`",
        link: env.BUILD_URL, result: currentBuild.currentResult,
        title: "Review-Service 배포",
        webhookURL: "https://discord.com/api/webhooks/1102411269262278738/V3K-UIBokEu1Gtj003xpuLuzwbRgaeIoDTioJRdG5uPRmtbg_3g0mxvCUYFb0voi0Am"
    }
}
}
}
}
}

```

10. Submit 클릭

Jenkins에 Gitlab 연동

- Multibranch Pipeline 프로젝트의 Configuration에 들어갔을 때, **Scan By webhook** 옵션에서 임의의 토큰 **dokki-trigger-token** 을 설정

Scan Multibranch Pipeline Triggers

- ☐ Periodically if not otherwise run ?
- ☒ Scan by webhook ?

Allows Multibranch Scan Webhook Trigger to trigger scan of this multibranch job.

(from [Multibranch Scan Webhook Trigger](#))

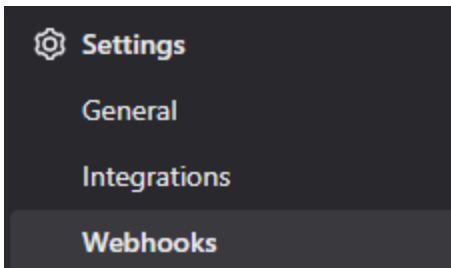
Trigger token ?

dokki-trigger-token

The token to match with webhook token. Receive any HTTP request, JENKINS_URL/multibranch-webhook-trigger/invoke?token=[Trigger token] If a token match, than a multibranch scan will bi triggered.

(from [Multibranch Scan Webhook Trigger](#))

2. Gitlab repo에서 왼쪽 네비바에서 Settings > Webhooks 클릭



3. URL에 아래처럼 token을 넣은 문자열을 입력하고, Push events를 추가한 후 Add Webhook 클릭

s08-final > S08P31E204 > Webhook Settings

Search page

Webhooks

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

URL

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

Secret token

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

Trigger

☒ Push events

Push to the repository.

Jenkins에 Docker hub Credential 정보 저장

1. Docker hub에 개인 계정으로 로그인 후, access Token을 발급
2. Dashboard > Manage Jenkins > Manage Credentials 클릭

Security



Configure Global Security
Secure Jenkins; define who is allowed to access/use the system.



Manage Credentials
Configure credentials



Configure Credential Providers
Configure the credential providers and types



Manage Users
Create/delete/modify users that can log in to this Jenkins.



In-process Script Approval
Allows a Jenkins administrator to review proposed scripts (written e.g. in Groovy) which run inside the Jenkins process and so could bypass security restrictions. **1 scripts pending approval.**

3. Credentials Add를 한 후, 아래처럼 작성 > Create

New credentials

Kind

Username with password

Scope ?

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

Username ?

noonmap

☐ Treat username as secret ?

Password ?

.....

ID ?

docker-hub-noonmap

Description ?

docker hub credential

Create

Jenkins Master/Slave 배포 구조 구성

1. master jenkins 프로젝트 내 셸에서 실행 > SSH 키 발급

```
sudo docker exec -it {jenkins 컨테이너 이름} /bin/bash
```

```
ssh-keygen -f ~/.ssh/jenkins_agent_key
```

2. Dashboard > Manage Jenkins > Credentials 에서 Credentials 추가 (SSH 인증)

Dashboard > Jenkins 관리 > Credentials

- 아까 발급받은 키를 `cat ~/.ssh/jenkins_agent_key` 로 조회해서 붙여넣기

New credentials

Kind

SSH Username with private key

Scope ?

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

ID ?

jenkins

Description ?

The jenkins ssh key

Username

jenkins

☐ Treat username as secret ?

Private Key

☒ Enter directly

Key

Enter New Secret Below

```
-----BEGIN OPENSSH PRIVATE KEY-----
b3wRwZ0CtP3mHnCO1T1TtWdW0XG0W3T00P4Z2B5T2VY5S1M1T1pJy0C8j9y000T90T3m00V6
NUiAvVITzzN7F0niUFpTeFIsEssZxD41Ps2Id8S8PF3ne1BB3AV0374KfcY0X19R4jCkIn
64cVOUXZhy3F/6t sTxg6l rwlqc7+obr3vqaj+XS1p18P7T6N6RQAjk i7IVUcK8kpkkcSP
ouBX1WhMpTNeFNAAAAEXJvb3RANTk2ZGJhZjAzZjk3AQ==
```

Passphrase







Create

⚠ 주의 : -----BEGIN OPENSSH PRIVATE KEY----- 이랑 -----END OPENSSH PRIVATE KEY----- 까지 모두 써줘야 함

Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
 dokki	jcw/*****	Username with password	
 docker-hub-noonmap	noonmap/***** (docker hub credential)	Username with password	docker hub credential 
 jenkins	jenkins (The jenkins ssh key)	SSH Username with private key	The jenkins ssh key 

3. public 키를 조회해서 복사해둔다.

```
cat ~/.ssh/jenkins_agent_key.pub
```

4. agent(slave) 서버에서 Jenkins agent 도커 컨테이너 실행 명령어를 실행한다.

- 22번 포트를 이미 사용하고 있어서 3000번 포트로 expose함
- `sudo ufw allow 3000` 방화벽을 오픈함.

```
docker run -d --rm --name=[컨테이너 명] -p [port번호]:22 \  
-e "JENKINS_AGENT_SSH_PUBKEY=[your-public-key]" \  
jenkins/ssh-agent:alpine
```

5. Dashboard > Manage Jenkins > Nodes > New Node해서 생성

New node

노드명

dev-server

Type

☒ Permanent Agent

Adds a plain, permanent agent to Jenkins. This is called "permanent" because Jenkins doesn't provide higher level of integration with these agents, such as dynamic provisioning. Select this type if no other agent types apply — for example such as when you are adding a physical computer, virtual machines managed outside Jenkins, etc.

☐ 노드복사

Create

?
1
jcw
로그아웃

Dashboard > Jenkins 관리 > Nodes > dev-server > Configure

상태

Delete Agent

설정

빌드 기록

통계보기

스크립트 콘솔

로그

시스템 정보

연결끊기

빌드 실행 상태

1 대기 중

Name ?

dev-server

Description ?

[Plain text] [미리보기](#)

Number of executors ?

1

Remote root directory ?

/home/jenkins

Labels ?

agent1

Usage ?

Use this node as much as possible

Launch method ?

Launch agents via SSH

Host ?

k8e204.p.ssafy.io

Credentials ?

jenkins (The jenkins ssh key)

Add +

Host Key Verification Strategy ?

Manually trusted key Verification Strategy

☐ Require manual verification of initial connection ?

고급

Edited

Availability ?

Keep this agent online as much as possible

Node Properties

☐ Disable deferred wipeout on this node ?

☐ Environment variables

☐ Tool Locations

Save

REST API Jenkins 2.387.2

- 이 때, 고급 > port 에서 아까 수정한 3000 번 포트로 수정해줘야 한다.

고급 ^ Edited

Port ?

3000

- 저장하고 난 후, 생성된 node를 클릭하고 → Launch agent 를 클릭해서 실행시킨다.
- 로그를 봤을 때 ssh connection error 어쩌고 하면서 fail이 뜬다.
 - 왼쪽 네비게이션 바에 나타난 Trust SSH Host Key 를 클릭한 후 Yes 를 눌러주자! trusted된 key가 아니라서 연결이 지연되는 거다.

상태
Delete Agent
설정
빌드 기록
통계보기
로그
Trust SSH Host Key
블루 오션 열기

```

SSHLauncher{host='k8e2041.p.ssafy.io', port=3000, credentialsId='prod-jenkins', jvmOptions='', javaPath='',
prefixStartSlaveCmd='', suffixStartSlaveCmd='', launchTimeoutSeconds=60, maxNumRetries=10, retryWaitTime=15,
sshHostKeyVerificationStrategy=udson.plugins.sshslaves.verifiers.ManuallyTrustedKeyVerificationStrategy,
tcpNoDelay=true, trackCredentials=true}
[05/12/23 14:04:56] [SSH] Opening SSH connection to k8e2041.p.ssafy.io:3000.
[05/12/23 14:04:56] [SSH] WARNING: The SSH key for this host is not currently trusted. Connections will be denied
until this new key is authorised.
Key exchange was not finished, connection is closed.
SSH Connection failed with IOException: "Key exchange was not finished, connection is closed.", retrying in 15
seconds. There are 10 more retries left.

```

- 성공하면 로그 길게 쪽쪽 나오다가 이렇게 뜬

```

Evacuated stdout
Agent successfully connected and online

```

Redis DB 추가

1. docker-compose.yml 파일을 아래와 같이 작성한 후, 저장

```

version: '3.5'

services:
  user-redis:
    image: redis:5.0.3
    container_name: user-redis
    hostname: user-redis
    networks:
      - dokki-network
    ports:
      - 6100:6379
    volumes:
      - /home/ubuntu/db/user-redis/data:/data
    command: redis-server --requirepass {설정할 비밀번호}
  timer-redis:
    image: redis:5.0.3
    container_name: timer-redis
    networks:
      - dokki-network
    ports:
      - 6200:6379
    volumes:
      - /home/ubuntu/db/timer-redis/data:/data
    command: redis-server --requirepass {설정할 비밀번호}

networks:
  dokki-network:
    external: true

```

2. `sudo docker-compose -f {파일이름} up -d` 명령어로 컨테이너를 실행

Mysql DB 추가

1. docker-compose.yml 파일을 아래와 같이 작성한 후, 저장

```

version: '3.5'

services:
  user-mysql:
    image: mysql:5.7
    container_name: user-mysql
    volumes:
      - /home/ubuntu/db/user-mysql/data:/var/lib/mysql
      #- /home/ubuntu/db/user-mysql/conf.d:/etc/mysql/conf.d # mysql config
      # - /home/ubuntu/db/user-mysql/sql:/bin/sql # sql files
      - /home/ubuntu/db/sql:/bin/sql # sql files
      - /home/ubuntu/db/conf.d:/etc/mysql/conf.d # mysql config

```

```

restart: always
command:
  - --character-set-server=utf8mb4
  - --collation-server=utf8mb4_unicode_ci
environment:
  - MYSQL_ROOT_PASSWORD=dokki44
  - MYSQL_DATABASE=dokki
  - TZ=Asia/Seoul
ports:
  - 7100:3306
networks:
  - dokki-network
timer-mysql:
  image: mysql:5.7
  container_name: timer-mysql
  volumes:
    - /home/ubuntu/db/timer-mysql/data:/var/lib/mysql # db
    #- /home/ubuntu/db/timer-mysql/conf.d:/etc/mysql/conf.d # mysql config
    #- /home/ubuntu/db/timer-mysql/sql:/bin/sql # sql files
    - /home/ubuntu/db/sql:/bin/sql # sql files
    - /home/ubuntu/db/conf.d:/etc/mysql/conf.d # mysql config
  restart: always
  command:
    - --character-set-server=utf8mb4
    - --collation-server=utf8mb4_unicode_ci
  environment:
    - MYSQL_ROOT_PASSWORD=dokki44
    - MYSQL_DATABASE=dokki
    - TZ=Asia/Seoul
  ports:
    - 7200:3306
  networks:
    - dokki-network
book-mysql:
  image: mysql:5.7
  container_name: book-mysql
  volumes:
    - /home/ubuntu/db/book-mysql/data:/var/lib/mysql
    #- /home/ubuntu/db/book-mysql/conf.d:/etc/mysql/conf.d # mysql config
    #- /home/ubuntu/db/book-mysql/sql:/bin/sql # sql files
    - /home/ubuntu/db/sql:/bin/sql # sql files
    - /home/ubuntu/db/conf.d:/etc/mysql/conf.d # mysql config
  restart: always
  command:
    - --character-set-server=utf8mb4
    - --collation-server=utf8mb4_unicode_ci
  environment:
    - MYSQL_ROOT_PASSWORD=dokki44
    - MYSQL_DATABASE=dokki
    - TZ=Asia/Seoul
  ports:
    - 7300:3306
  networks:
    - dokki-network

```



```

review-mysql:
  image: mysql:5.7
  container_name: review-mysql
  volumes:
    - /home/ubuntu/db/review-mysql/data:/var/lib/mysql
    #- /home/ubuntu/db/review-mysql/conf.d:/etc/mysql/conf.d # mysql config
    #- /home/ubuntu/db/review-mysql/sql:/bin/sql # sql files
    - /home/ubuntu/db/sql:/bin/sql # sql files
    - /home/ubuntu/db/conf.d:/etc/mysql/conf.d # mysql config
  restart: always
  command:
    - --character-set-server=utf8mb4
    - --collation-server=utf8mb4_unicode_ci
  environment:
    - MYSQL_ROOT_PASSWORD=dokki44
    - MYSQL_DATABASE=dokki
    - TZ=Asia/Seoul
  ports:
    - 7400:3306
  networks:
    - dokki-network
networks:
  dokki-network:
    external: true

```

2. `/home/ubuntu/db/conf.d` 경로에 `mysql_conf.cnf` 파일 생성 (해당 경로에 디렉토리 생성)

```

[mysqld]
character-set-server = utf8
collation-server = utf8_unicode_ci
skip-character-set-client-handshake

```

3. `sudo docker-compose -f {파일이름} up -d` 명령어로 컨테이너를 실행
4. volume으로 지정한 `/bin/sql`에 아래와 같은 `create-user.sql` 파일을 생성
 - ip를 `"172.18.%"` 으로 하면 docker network 내에서? 만 접속이 가능함

```

create user if not exists dokki@'172.18.%' identified by '설정할 비밀번호';
-- dokki 데이터베이스의 모든 테이블에 대한 권한 부여
grant all privileges on dokki.* to dokki@'172.18.%;
-- 변경된 내용을 메모리에 반영 (권한 적용)
flush privileges;

```

5. 아래의 명령어로 mysql 컨테이너들마다 sql 파일을 실행시킴

```
sudo docker exec -it user-mysql mysql -p{비밀번호} -e "source /bin/sql/create-user.sql"
```

환경 변수

- **DB_URL**
연결될 MySQL 데이터베이스의 URL 정보
- **DB_USER**
연결될 MySQL 데이터베이스의 계정
- **DB_PASSWD**
연결될 MySQL 데이터베이스의 비밀번호
- **PORT**
서비스가 expose될 port 번호
- **REDIS_HOST**
연결될 Redis 데이터베이스의 네트워크 ip 주소
- **REDIS_PORT**
연결될 Redis 데이터베이스의 컨테이너 내부 네트워크 port
- **REDIS_PASSWORD**
연결될 Redis 데이터베이스의 비밀번호
- **UPLOAD_PATH**
파일이 업로드 되는 절대 경로
- **UPLOAD_DIR**
파일 업로드 디렉토리 (UPLOAD_PATH 아래에 있는 디렉토리)
- **HOST_URI**
서버 URL (https://dokki.kr)
- **OPENAI_API_TOKEN**
ChatGPT와 DALL-E2 API를 사용할 때 필요한 API Token


외부 서비스 활용

OpenAI 가입 및 활용 방법

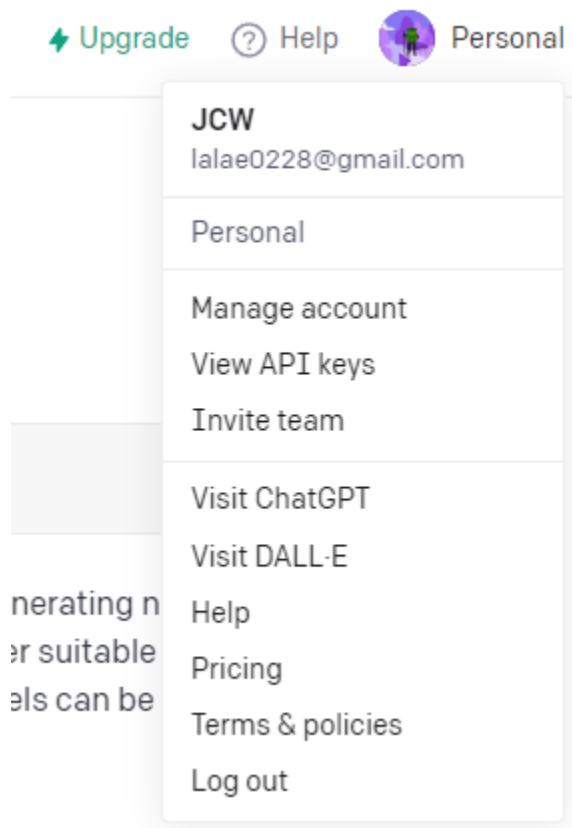
1. 다음 사이트에서 회원가입 후 로그인을 한다

OpenAI API

An API for accessing new AI models developed by OpenAI

 <https://platform.openai.com/docs/introduction>

2. 우측 상단의 계정 아이콘을 클릭 > “View API keys” 를 클릭



3. Create new secret key를 클릭

API keys

Your secret API keys are listed below. Please note that we do not display your secret API keys again after you generate them.

Do not share your API key with others, or expose it in the browser or other client-side code. In order to protect the security of your account, OpenAI may also automatically rotate any API key that we've found has leaked publicly.

NAME	KEY	CREATED	LAST USED ⓘ	
DokkiSecretKey	sk-...5b4c	2023년 5월 3일	2023년 5월 4일	 
+ Create new secret key				

4. 임의의 문자를 입력 후 key를 생성

Create new secret key


Name Optional

CancelCreate secret key

알라딘 API 활용 방법

1. 다음의 알라딘 사이트에서 회원가입 진행

[알라딘] "좋은 책을 고르는 방법, 알라딘"

 https://www.aladin.co.kr/login/wlogin.aspx?returnurl=https://www.aladin.co.kr/ttb/wblog_manage.aspx

2. 사용 URL을 입력한 후, API를 발급




Kakao Login API 활용 방법

1. 다음 사이트에서 회원가입 및 로그인을 한다

Kakao Developers

카카오 API를 활용하여 다양한 어플리케이션을 개발해보세요. 카카오톡 로그인, 메시지 보내기, 친구 API, 인공지능 API 등을 제공합니다.

 <https://developers.kakao.com/>

kakao developers

2. 디버그, 릴리즈, 스토어 서명키를 해시키로 변환하여 등록

Android 플랫폼 수정

패키지명 [필수]

Android 앱의 고유 값입니다. AndroidManifest.xml의 package= 어트리뷰트를 참조하세요

com.example.dokki

마켓 URL

없음



키 해시

키 해시가 등록된 앱만 SDK를 이용해 API를 호출할 수 있습니다.

여러 개의 키 해시는 줄바꿈으로 추가하세요.

취소

저장

3. 디버그 해시키 발급 방법



[WINDOW]

```
keytool -exportcert -alias androiddebugkey -keystore C:\Users\[사용자명]\.android\debug.keystore -storepass android -keypass android | openssl sha1 -binary | openssl base64
```

[MAC]

```
keytool -exportcert -alias androiddebugkey -keystore ~/.android/debug.keystore -storepass android -keypass android | openssl sha1 -binary | openssl base64
```

4. 릴리즈 해시키 발급 방법

- openssl 설치 , Git 설치 必



[앱 서명 파일 생성(.jks)]

[MAC]

```
keytool -genkey -v -keystore ~/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

[WINDOW]

```
keytool -genkey -v -keystore c:/Users/USER_NAME/key.jks -keyalg RSA -keysize 2048 -validity 10000 -alias key
```

[릴리즈 해시키 발급]

RELEASE_KEY_ALIAS : 앱 서명 키의 별칭

RELEASE_KEY_PATH : 발급한 .jks 파일의 위치

[MAC]

```
keytool -exportcert -alias <RELEASE_KEY_ALIAS> -keystore <RELEASE_KEY_PATH> | openssl sha1 -binary | openssl base64
```

[WINDOW]

```
keytool -exportcert -alias <RELEASE_KEY_ALIAS> -keystore <RELEASE_KEY_PATH> | openssl sha1 -binary | PATH_TO_OPENSSL_LIBRARY\bin\openssl base64
```

5. 마켓 앱 서명키 발급 방법

서명키

[다운로드](#)

MD5 인증서 지문

SHA-1 인증서 지문

SHA-256 인증서 지문



[마켓의 SHA-1 인증서와 SHA-256 인증서를 해시키로 변환]

리눅스 환경의 터미널에서만 가능

```
echo [SHA-1 인증서 OR SHA-256 인증서] | xxd -r -p | openssl base64
```

EX)

```
echo 33:4E:48:84:19:50:3A:1F:63:A6:0F:F6:A1:C2:31:E5:01:38:55:2E | xxd -r -p |  
openssl base64
```