

Jenkins Blue Ocean 빌드/배포 자동화



본 문서는 Jenkins 서버와 deploy 서버가 같은 클라우드 서버 내에 있습니다.

Jenkins를 통한 수동 배포

1. deploy 서버에 Docker 설치 & Gitlab Repository Clone

- 다음 명령어를 실행한다.

```
$ sudo apt update
$ sudo apt upgrade
```

- 다음 패키지들을 설치한다.

```
$ sudo apt install apt-transport-https ca-certificates
$ sudo apt install curl gnupg-agent software-properties-common
```

- Docker의 공식 GPG 키를 추가한다

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

- `stable repository` 를 세팅하기 위한 명령어를 실행한다.

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

- 가장 최신 버전의 Docker 엔진을 설치한 후, 버전을 확인한다.

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

- Gitlab Repository를 클론한다.

```
$ git clone https://<your-repository>.git
```

2. Docker 컨테이너에 Jenkins 설치 및 진행

- Docker 컨테이너에 Jenkins 설치 후 구동

```
$ sudo docker run -d \
-u root \
-p 9090:8080 \
--name=jenkins \
-v /home/ubuntu/docker/jenkins-data:/var/jenkins_home \
-v /var/run/docker.sock:/var/run/docker.sock \
-v "$HOME":/home/jenkinsci/blueocean \
jenkinsci/blueocean
```

- 또는 docker-compose.yml 작성

```
version: '3.7' # 도커 버전에 맞는 버전으로 작성한다.

services:
  jenkins:
    image: 'jenkinsci/blueocean'
    restart: unless-stopped
    user: root
    privileged: true
    ports:
      - '9090:8080'
    volumes:
      - '/home/ubuntu/docker/jenkins-data:/var/jenkins_home'
      - '/var/run/docker.sock:/var/run/docker.sock'
      - '$HOME:/home'
    container_name: 'jenkins'
```



-u root와 privileged: true 옵션으로 실행하지 않으면 추후 진행시 에러가 발생한다.



volume에 해당하는 폴더가 host의 root 권한이 있어야 접근 가능하기 때문이다.

- 도커 컨테이너 접속

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

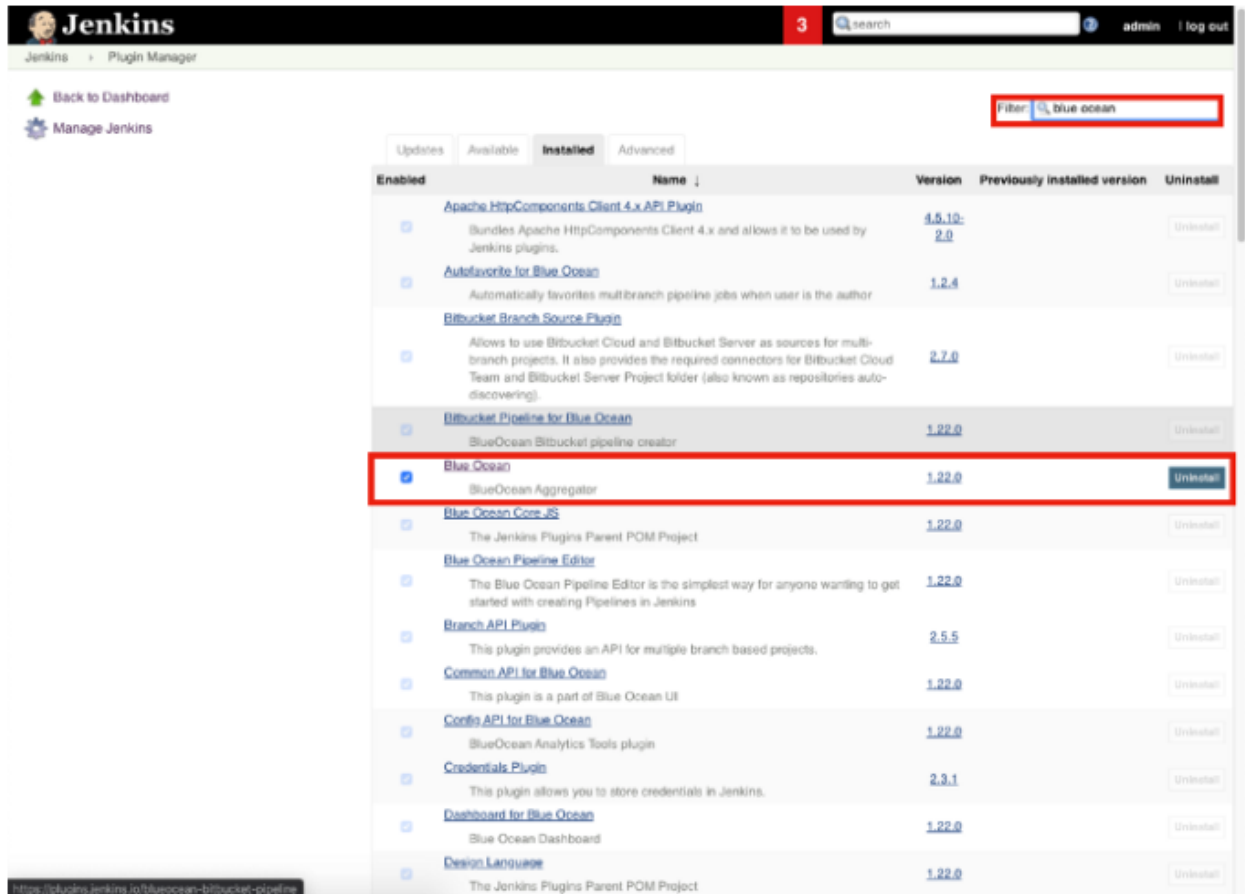
- http://<your-aws-domain>:<jenkins port> 접속 후 admin password 입력

```
$ cat /var/jenkins_home/secrets/initialAdminPassword
```

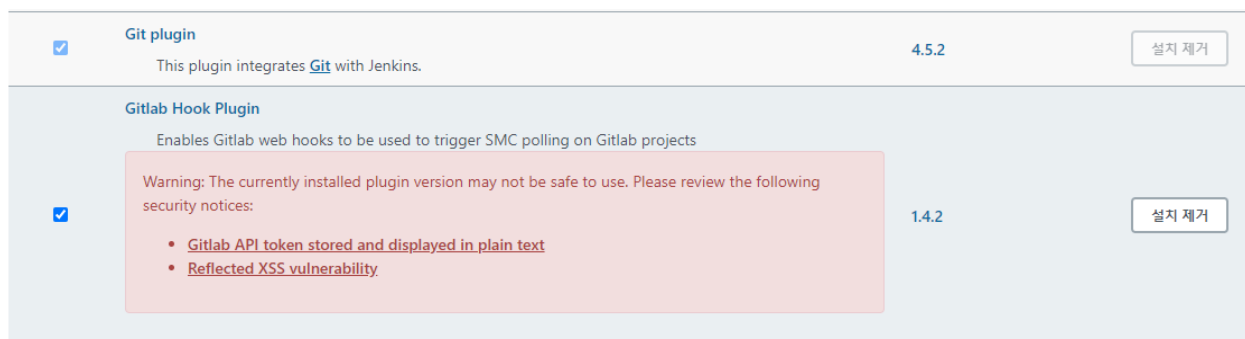
- jenkins 설치 (install suggested plugin)

3. Jenkins 플러그인 설치 및 환경설정

- Jenkins blue ocean 설치



- Gitlab 설치



- jenkins 관리 → 시스템 설정에서 gitlab 관련 설정 추가

Gitlab

Enable authentication for '/project' end-point ☒

GitLab connections

Connection name

A name for the connection

Gitlab host URL

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

Credentials


API Token for accessing Gitlab

- Credentials는 Gitlab에서 발급받은 API token 입력


Add Credentials


Domain

Kind

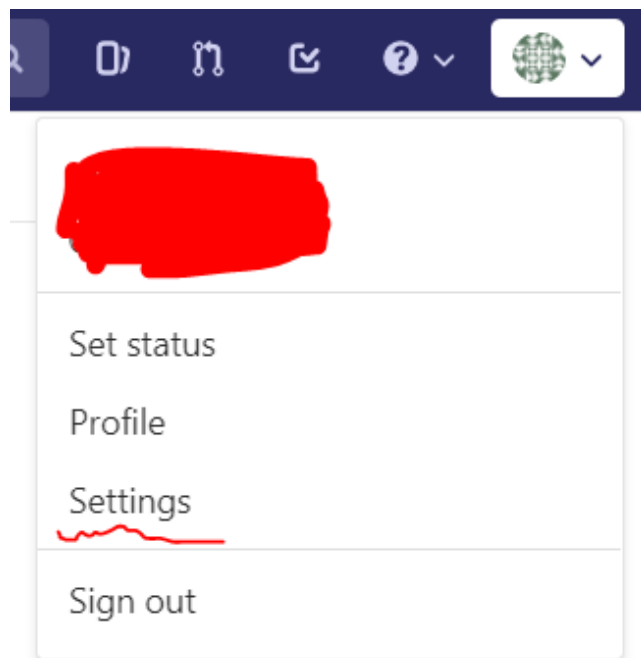
Scope 

API token

ID 

Description 

- 생성 시 발급된 토큰은 복사 후 로컬 PC에 저장



User Settings

Profile

Account

Applications

Chat

Access Tokens

Emails

Password

Notifications

SSH Keys

GPG Keys

Preferences

Active Sessions

User Settings > Access Tokens

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.

Add a personal access token

Pick a name for the application, and we'll give you a unique personal access token.

Name

ssafy_coach

Expires at

YYYY-MM-DD

Scopes

☒ api

Grants complete read/write access to the API, including all groups and projects.

☐ read_user

Grants read-only access to the authenticated user's profile through the /user API endpoint, which includes username, public email, and full name. Also grants access to read-only API endpoints under /users.

☐ read_repository

Grants read-only access to repositories on private projects using Git-over-HTTP (not using the API).

Create personal access token

- Jenkins Location 입력

Jenkins Location

Jenkins URL

http://[redacted].ssafy.io:9090/

System Admin e-mail address

address not configured yet <nobody@nowhere>

4. Jenkins와 Gitlab Repository 연결

- 새로운 item → pipeline 선택

Enter an item name

» Required field



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, 플랫폼 특성 빌드, 기타 등등 처럼 다수의 서로다른 환경설정이 필요한 프로젝트에 적합함.



Bitbucket Team/Project

Scans a Bitbucket Cloud Team (or Bitbucket Server Project) for all repositories matching some defined markers.



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.



GitHub Organization

Scans a GitHub organization (or user account) for all repositories matching some defined markers.



Multibranch Pipeline

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

- Jenkins pipeline 설정 입력
- Repository URL은 Gitlab Repository URL 입력
- Credentials는 ADD 한 후 Secret Text 타입으로 변경 후 gitlab id와 위에서 발급받은 token 추가
- script path는 Jenkinsfile이 존재하는 폴더 위치

General Build Triggers Advanced Project Options **Pipeline**

Pipeline

Definition Pipeline script from SCM

SCM Git

Repositories

Repository URL [redacted]

Credentials [redacted] Add

고급...

Add Repository

Branches to build

Branch Specifier (blank for 'any') */develop

Add Branch

Repository browser (자동)

Additional Behaviours Add

Script Path ./jenkins/jenkinsfile

5. Gitlab 디렉토리에 Dockerfile 생성

- backend Dockerfile 생성

```
total 48
drwxrwxr-x  4 ubuntu ubuntu 4096 Jan 27 13:35 .
drwxrwxr-x 10 ubuntu ubuntu 4096 Jan 27 13:35 ..
-rw-rw-r--  1 ubuntu ubuntu  395 Jan 26 13:25 .gitignore
drwxrwxr-x  3 ubuntu ubuntu 4096 Jan 26 13:25 .mvn
-rw-rw-r--  1 ubuntu ubuntu  145 Jan 27 13:29 Dockerfile
-rw-rw-r--  1 ubuntu ubuntu 10070 Jan 26 13:25 mvnw
-rw-rw-r--  1 ubuntu ubuntu  6608 Jan 26 13:25 mvnw.cmd
-rw-rw-r--  1 ubuntu ubuntu  3633 Jan 26 13:27 pom.xml
drwxrwxr-x  3 ubuntu ubuntu 4096 Jan 26 13:25 src
```

- backend Dockerfile 작성

```
FROM openjdk:8-jdk-alpine
VOLUME /tmp
ADD ./target/<pom.xml-artifactID>-0.0.1-SNAPSHOT.jar app.jar
ENV JAVA_OPTS=""
ENTRYPOINT ["java", "-jar", "/app.jar"]
```

- frontend Dockerfile 생성


```
total 92
drwxrwxr-x 6 ubuntu ubuntu 4096 Jan 27 13:29 .
drwxrwxr-x 10 ubuntu ubuntu 4096 Jan 27 13:35 ..
-rw-rw-r-- 1 ubuntu ubuntu 32 Jan 26 13:25 .browserslistrc
-rw-rw-r-- 1 ubuntu ubuntu 68 Jan 26 13:27 .env.development
-rw-rw-r-- 1 ubuntu ubuntu 401 Jan 27 11:51 .eslintrc.js
drwxrwxr-x 3 ubuntu ubuntu 4096 Jan 26 13:27 .github
-rw-rw-r-- 1 ubuntu ubuntu 238 Jan 26 13:25 .gitignore
-rw-rw-r-- 1 ubuntu ubuntu 60 Jan 26 13:25 .postcssrc.js
-rw-rw-r-- 1 ubuntu ubuntu 647 Jan 26 13:25 CHANGELOG.md
-rw-rw-r-- 1 ubuntu ubuntu 380 Jan 27 13:29 Dockerfile
-rw-rw-r-- 1 ubuntu ubuntu 385 Jan 26 13:25 ISSUE_TEMPLATE.md
-rw-rw-r-- 1 ubuntu ubuntu 1069 Jan 26 13:25 LICENSE.md
-rw-rw-r-- 1 ubuntu ubuntu 13412 Jan 26 13:25 README.md
-rw-rw-r-- 1 ubuntu ubuntu 46 Jan 26 13:25 babel.config.js
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 27 13:35 nginx
-rw-rw-r-- 1 ubuntu ubuntu 1491 Jan 26 13:27 package.json
-rw-rw-r-- 1 ubuntu ubuntu 1474 Jan 26 13:28 public
drwxrwxr-x 2 ubuntu ubuntu 4096 Jan 26 13:27 src
drwxrwxr-x 10 ubuntu ubuntu 4096 Jan 26 13:27 vue.config.js
```

- frontend Dockerfile 작성

```
FROM node:lts-alpine as build-stage
WORKDIR /homepage
COPY package*.json ./

RUN npm install
COPY . .
RUN npm run build

FROM nginx:stable-alpine as production-stage
RUN rm /etc/nginx/conf.d/default.conf
COPY ./nginx/homepage.conf /etc/nginx/conf.d/homepage.conf

COPY --from=build-stage ./homepage/dist /usr/share/nginx/html/homepage
EXPOSE 80
CMD ["nginx", "-g", "daemon off;"]
```

- Jenkinsfile 생성

```
total 12
drwxrwxr-x  2 ubuntu ubuntu 4096 Jan 27 13:59 .
drwxrwxr-x 10 ubuntu ubuntu 4096 Jan 27 13:35 ..
-rw-rw-r--  1 ubuntu ubuntu 1429 Jan 27 13:57 Jenkinsfile
```

- Jenkinsfile 작성
- 주석은 작성하지 않아도 됩니다

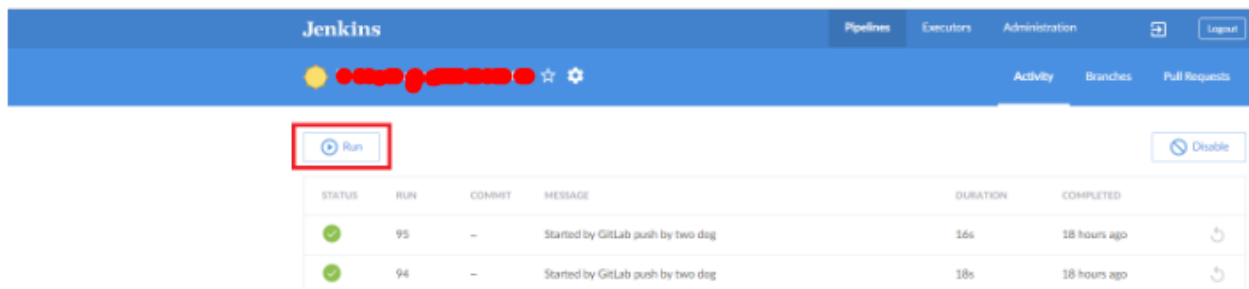
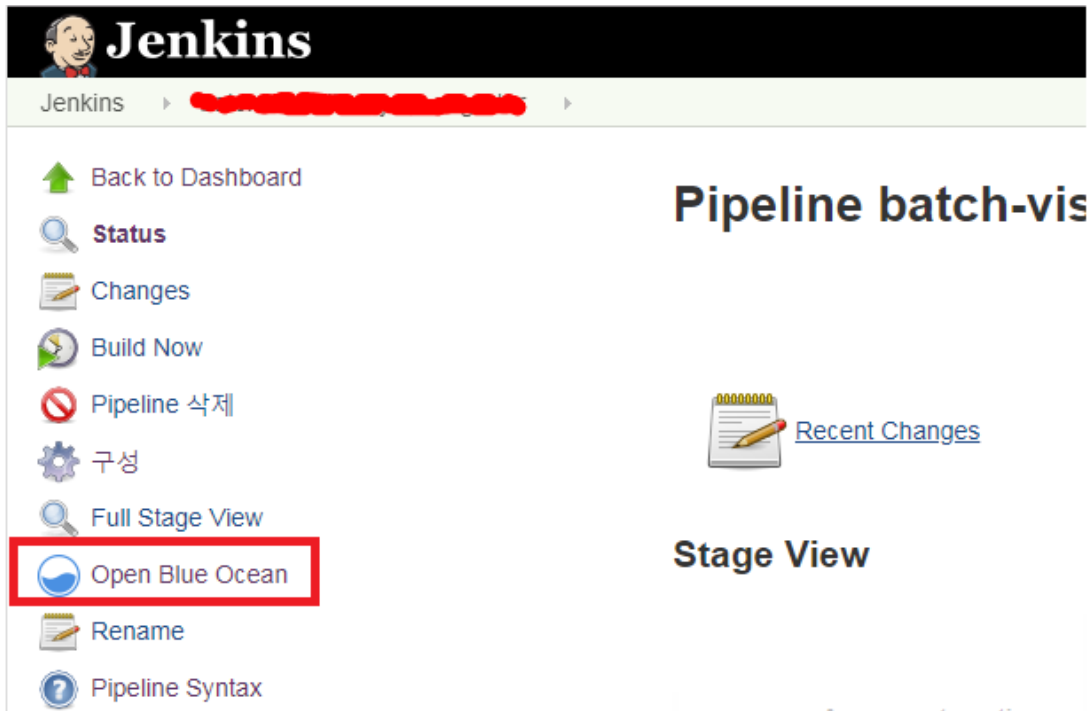
```
// 젠킨스 파이프라인 플러그인을 호출하기 위한 블록
pipeline {
    // 파이프라인을 실행하고 싶은 위치 정의
    agent none
    // gitlab의 소스를 jenkins 디렉토리로 내려받을 시
    // skipDefaultCheckout(true)일 경우 내려받는 프로세스 skip
    // skipDefaultCheckout(false)일 경우 gitlab 소스 체크
    options { skipDefaultCheckout(true) }
    // stage의 모음
    stages {
        // 실제 작업이 수행되는 블록
        // 해당 stage 명으로 jenkins 화면에 표시된다
        stage('Build and Test') {
            // docker image에 명시된 image를 활용하여 steps 수행
            agent {
                docker {
                    image 'maven:3-alpine'
                    args '-v /root/.m2:/root/.m2'
                }
            }
            options { skipDefaultCheckout(false) }
            steps {
                sh 'mvn -B -DskipTests -f <your-pom.xml-directory> clean package'
            }
        }
        stage('Docker build') {
            agent any
            steps {
                sh 'docker build -t <front-image-name>:latest <front dockerfile path>'
                sh 'docker build -t <back-image-name>:latest <back dockerfile path>'
            }
        }
        stage('Docker run') {
            agent any
            steps {
                // 현재 동작중인 컨테이너 중 <front-image-name>의 이름을 가진
                // 컨테이너를 stop 한다
                sh 'docker ps -f name=<front-image-name> -q \
                | xargs --no-run-if-empty docker container stop'
                // 현재 동작중인 컨테이너 중 <back-image-name>의 이름을 가진
                // 컨테이너를 stop 한다
                sh 'docker ps -f name=<back-image-name> -q \
                | xargs --no-run-if-empty docker container stop'
            }
        }
    }
}
```

```

// <front-image-name>의 이름을 가진 컨테이너를 삭제한다.
sh 'docker container ls -a -f name=<front-image-name> -q \
  | xargs -r docker container rm'
// <back-image-name>의 이름을 가진 컨테이너를 삭제한다.
sh 'docker container ls -a -f name=<back-image-name> -q \
  | xargs -r docker container rm'
// docker image build 시 기존에 존재하던 이미지는
// dangling 상태가 되기 때문에 이미지를 일괄 삭제
sh 'docker images -f dangling=true && \
  docker rmi $(docker images -f "dangling=true" -q)'
// docker container 실행
sh 'docker run -d --name <front-image-name> -p 80:80 <front-image-name>:latest'
sh 'docker run -d --name <back-image-name> -p 8080:8080 <back-image-name>:latest'
}
}
}
}

```

6. 소스코드 Push 후 수동 배포



7. Jenkins Build Triggers 설정

- Build when a change is pushed to Gitlab webhook 체크

- Build Triggers 고급... 클릭 후
- include에서 webhook 브랜치 선택
- 우측 아래 Generate 클릭

8. Gitlab에서 Webhook 설정

- 프로젝트에서 Settings → Integrations 선택
- URL은 Build Triggers 설정 시 보였던 Gitlab Webhook URL 입력
- Secret token은 Build Triggers 설정 시 생성했 던 Secret Token 입력

Integrations

Webhooks can be used for binding events when something is happening within the project.

URL

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

- Webhook 테스트 후 200 리턴 시 정상 작동

☐ Job events

This URL will be triggered when the job status changes

☐ Pipeline events

This URL will be triggered when the pipeline status changes

☐ Wiki Page events

This URL will be triggered when a wiki page is created/updated

SSL verification

☒ Enable SSL verification

Add webhook

Webhooks (1)

http://[redacted]
[redacted]

Push Events

SSL Verification: enabled

Edit

Test



Push events

Tag push events

Issues events

Service	Description
Asana	Asana - Teamwork without

Hook executed successfully: HTTP 200

Integrations

[Webhooks](#) can be used for binding events when something is happening within the project.

URL

Secret Token

Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.

Trigger

☒ **Push events**

This URL will be triggered by a push to the repository