



# 포팅 매뉴얼

## ▼ Contents

### 1. Version

[Front-end](#)

[Back-end](#)

### 2. 외부 API

[Front-end](#)

[Back-end](#)

### 3. Deploy

[EC2 환경 세팅](#)

[Docker 설치](#)

[docker-compose 설치](#)

[Jenkins 설정](#)

[Pipeline 설정](#)

✓ [Back-end pipeline script](#)

✓ [Front-end pipeline script](#)

[Docker에 MySQL 설치 & 실행](#)

[MySQL bash 접속, 계정 생성](#)

[Nginx 설치](#)

## 1. Version

### Front-end

- Node.js: 18.15.0
- TypeScript: 5.0.4
- @types/node: 18.16.0
- @types/react: ^18.2.6
- next: 13.3.1
- next-pwa: ^5.6.0
- @tanstack/react-query: ^4.29.3
- jotai: ^2.0.4
- axios: ^1.3.6
- @emotion/react: ^11.10.6

### Back-end

- IntelliJ Ultimate: 2022.3.1
- Java: 11
- Spring boot 2.7.11
- gradle 7.6.1
- MySQL: 8.0.29
- MongoDB: 5.0.17
- Docker: 23.0.4
- Docker-compose: 1.29.2
- Jenkins: 2.387.2
- nginx: 1.18.0

## 2. 외부 API

### Front-end

### Back-end

- 

- AWS S3
- AWS CloudFront

#### ▼ Front-end next.config.js

```
/** @type {import('next').NextConfig} */
const withPWA = require("next-pwa")({
  dest: "public",
  disable: process.env.NODE_ENV === "development",
})

module.exports = withPWA({
  reactStrictMode: false,
  swcMinify: true,
  compiler: {
    emotion: true,
  },
  images: {
    domains: ["placeholder.it", "placeimg.com", "d3bkfkkihvj5ql.cloudfront.net"],
  },
  async rewrites() {
    return [
      {
        source: "/api/:path*",
        destination: "https://k8d103.p.ssafy.io/api/:path*",
      },
    ];
  },
});

module.exports = {
  ...module.exports,
  compilerOptions: {
    target: "es2015",
  },
  async rewrites() {
    return [
      {
        source: "/api/:path*",
        destination: "https://k8d103.p.ssafy.io/api/:path*",
      },
    ];
  },
}
```

#### ▼ Front-end tsconfig.json

```
{
  "compilerOptions": {
    "target": "es5",
    "lib": ["dom", "dom.iterable", "esnext"],
    "allowJs": true,
    "skipLibCheck": true,
    "strict": true,
    "forceConsistentCasingInFileNames": true,
    "noEmit": true,
    "esModuleInterop": true,
    "module": "esnext",
    "moduleResolution": "node",
    "resolveJsonModule": true,
    "isolatedModules": true,
    "jsx": "preserve",
    "jsxImportSource": "@emotion/react",
    "incremental": true,
    "paths": {
      "@/*": [".src/*"]
    }
  },
}
```

```
"include": ["next-env.d.ts", "**/*.ts", "**/*.tsx"],
"exclude": ["node_modules"]
}
```

## ▼ Front-end swcrc

```
{
  "jsc": {
    "experimental": {
      "plugins": [["@swc-jotai/react-refresh", {}]]
    }
  },
}
```

## ▼ Back-end build.gradle

```
plugins {
    id 'java'
    id 'org.springframework.boot' version '2.7.11'
    id 'io.spring.dependency-management' version '1.0.15.RELEASE'
}

repositories {
    mavenCentral()
}

bootJar.enabled = false

subprojects {
    group = 'com.ico'
    version = '0.0.1-SNAPSHOT'
    sourceCompatibility = '11'
    compileJava.options.encoding = 'UTF-8'

    apply plugin: 'java'
    // build.gradle에서 api() 를 사용하려면 java-library 사용
    apply plugin: 'java-library'
    apply plugin: 'org.springframework.boot'
    // spring boot dependency를 사용하여 사용중인 부트 버전에서 자동으로 의존성을 가져온다.
    apply plugin: 'io.spring.dependency-management'

    configurations {
        compileOnly {
            extendsFrom annotationProcessor
        }
    }

    repositories {
        mavenCentral()

        flatDir {
            dirs 'libs'
        }
    }

    // 관리하는 모듈에 공통 dependencies
    dependencies {
        implementation 'org.springframework.boot:spring-boot-starter-web'
        implementation 'org.springframework.boot:spring-boot-starter-data-jpa'
        implementation 'org.springframework.boot:spring-boot-starter-data-mongodb'
        implementation 'org.springframework.boot:spring-boot-starter-validation'
        compileOnly 'org.projectlombok:lombok'
        developmentOnly 'org.springframework.boot:spring-boot-devtools'
        runtimeOnly 'com.h2database:h2'
        annotationProcessor 'org.springframework.boot:spring-boot-configuration-processor'
        annotationProcessor 'org.projectlombok:lombok'
        testImplementation 'org.springframework.boot:spring-boot-starter-test'
        testImplementation 'junit:junit:4.13.1'
        testImplementation 'org.junit.jupiter:junit-jupiter-api:5.8.1'
        testRuntimeOnly 'org.junit.jupiter:junit-jupiter-engine:5.8.1'
        implementation 'mysql:mysql-connector-java:8.0.29'
        // Spring Security v-5.6.4
        implementation 'org.springframework.boot:spring-boot-starter-security'
```

```

// JWT
implementation 'io.jsonwebtoken:jjwt-api:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-impl:0.11.2'
runtimeOnly 'io.jsonwebtoken:jjwt-jackson:0.11.2'
// https://mvnrepository.com/artifact/net.nurigo/sdk
implementation group: 'net.nurigo', name: 'sdk', version: '4.2.7'

}

test {
    useJUnitPlatform()
}
}

project(":core-module") {
    bootJar.enabled = false
    jar.enabled = true

    dependencies {
        runtimeOnly 'org.mariadb.jdbc:mariadb-java-client'
        // 테스트 내장형 몽고DB 의존성
        testImplementation 'de.flapdoodle.embed:de.flapdoodle.embed.mongo'
    }
}

project(":api-module") {

    bootJar {
        archiveBaseName.set('ico')
        archiveFileName.set("ico-api-module-0.0.1.jar")
        archiveVersion.set("0.0.1")
    }

    dependencies {
        implementation fileTree(dir: 'libs', include: ['*.jar'])
        // AWS
        implementation 'org.springframework.cloud:spring-cloud-starter-aws:2.2.6.RELEASE'
        implementation project(':core-module')
    }
}

project(":batch-module") {

    bootJar {
        archiveBaseName.set('ico')
        archiveFileName.set("ico-batch-module-0.0.1.jar")
        archiveVersion.set("0.0.1")
    }

    dependencies {
        implementation fileTree(dir: 'libs', include: ['*.jar'])
        implementation 'org.springframework.boot:spring-boot-starter-batch'
        testImplementation 'org.springframework.batch:spring-batch-test'
        implementation project(':core-module')
    }
}
}

```

## ▼ Back-end yml files

```

resources
├ application-aws.yml
├ application-login.yml
├ application-db.yml
└ application.yml

```

```

resources
├ application-login.yml
├ application-db.yml
└ application.yml

```

```

cloud:
  aws:

```

```
s3:
  bucket: {S3 버킷 이름}
  credentials:
    access-key:
    secret-key:
  region:
    static: {지역}
  stack:
    auto: false

cloud-front:
  domain: {CloudFront 도메인}
```

```
spring:
  # Security 및 JWT 설정
  security:
    securityKey: "{암호화할 비밀 키}"
    jwt:
      header: Authorization
      # linux 환경에서 아래 코드 입력하면 키 생성됨
      # echo 'silvernine-tech-spring-boot-jwt-tutorial-secret-silvernine-tech-spring-boot-jwt-tutorial-secret'|base64
      secret: {암호화할 비밀 키}
      token-validity-in-seconds: 2592000

  coolsms:
    apiKey:
    apiSecret:
    fromPhoneNum: {메세지를 전송하는 번호}
```

```
# 공통 설정
spring:
  data:
    mongodb:
      uri: mongodb+srv://{이름}:{비밀번호}@ssafy.ngivl.mongodb.net/S08P31D103
      authentication-database: admin

--- # dev 설정
spring:
  config:
    activate:
      on-profile: db-dev

  datasource:
    url: jdbc:mysql://k8d103.p.ssafy.io:3306/ico?serverTimezone=Asia/Seoul
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: {이름}
    password: {비밀번호}
    hikari:
      connection-timeout: 58000
      max-lifetime: 580000

--- # local 설정
spring:
  config:
    activate:
      on-profile: db-local

  datasource:
    url: jdbc:mariadb://stg-yswa-kr-practice-db-master.mariadb.database.azure.com:3306/S08P31D103
    driver-class-name: org.mariadb.jdbc.Driver
    username: {이름}
    password: {비밀번호}
    hikari:
      connection-timeout: 58000
      max-lifetime: 580000
```

```
logging:
  level:
    root: warn
    com.ico: info
    com.ico.api: info
    org.hibernate.type.descriptor.sql: trace
```

```

com.amazonaws.util.EC2MetadataUtils: error
# 로컬 실행 시 aws 환경이 아니라 에러 로그 출력된다.
# 배포하여 aws 환경에서는 문제가 없기 때문에 실행 시 로그를 감춘다.

server:
# address: 192.168.100.166 프론트와 로컬 연결
port: 8081
# domain 설정 시 로컬로 실행하여 같은 네트워크에서 해당 주소로 접근할 수 있다.
# domain: http://127.0.0.1
servlet:
  session:
    timeout: 1440m
  max-http-header-size: 3145728

spring:
  profiles:
    active: dev
    group:
      dev:
        - db-dev
      local:
        - db-local
  include:
    - db
    - login
    - aws

  jpa:
    database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
    generate-ddl: true
    hibernate:
      # create(new table), update(add new column), none(nothing)
      ddl-auto: update
    properties:
      hibernate:
        format_sql: true
        show_sql: true
        use_sql_comments: true
      open-in-view: false
      defer-datasource-initialization: true
  sql:
    init:
      mode: always
      continue-on-error: true

  servlet:
    multipart:
      enabled: true
      max-file-size: 100MB
      max-request-size: 100MB

  dependencies:
    net.nurigo:
      java-sdk: 3.5

```

```

logging:
  level:
    root: warn
    com.ico: info
    com.ico.api: info
    org.hibernate.type.descriptor.sql: trace

server:
  port: 8082
  # domain: http://127.0.0.1
  servlet:
    session:
      timeout: 1440m
    max-http-header-size: 3145728

spring:
  profiles:
    active: dev
    group:
      dev:
        - db-dev
      local:

```

```

- db-local
include:
- db
- login

batch:
job:
  names: ${job.name:NONE}
  enabled: false
jdbc:
  initialize-schema: always
jpa:
  database-platform: org.hibernate.dialect.MySQL5InnoDBDialect
  generate-ddl: true
  hibernate:
    # create(new table), update(add new column), none(nothing)
    ddl-auto: update
  properties:
    hibernate:
      format_sql: true
      show_sql: true
      use_sql_comments: true
  open-in-view: false
  defer-datasource-initialization: true
sql:
  init:
    mode: always
    continue-on-error: true

```

## 3. Deploy

### EC2 환경 세팅

| MobaXterm 프로그램 사용

### Docker 설치

✓ docker 이전 버전 제거

```
sudo apt-get remove docker docker-engine docker.io containerd runc
```

✓ 패키지 업데이트 및 apt 패키지 설치

```

$ sudo apt-get update

$ sudo apt-get install \
  ca-certificates \
  curl \
  gnupg \
  lsb-release

```

✓ Docker의 공식 GPG 키를 추가

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

### ✓ 리포지토리 설정

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

### ✓ docker 엔진 설치

```
$ sudo apt-get update
```

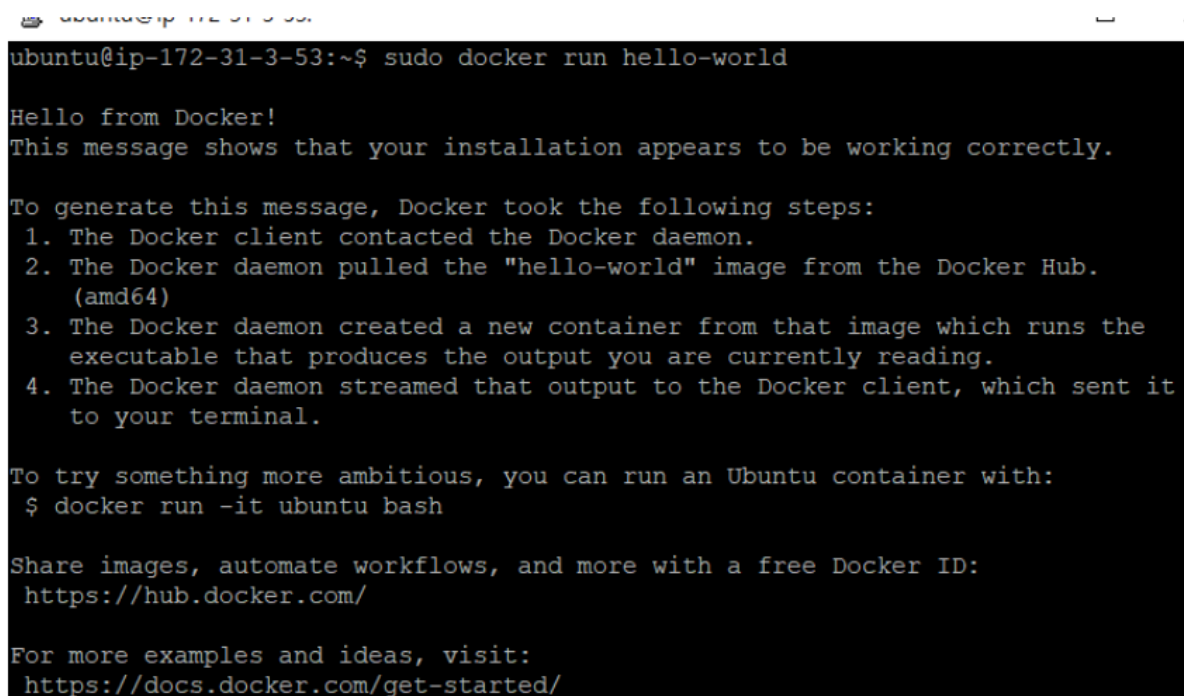
### ✓ docker 최신 버전 설치

```
$ sudo apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

### ✓ hello-world 이미지를 실행하여 Docker 엔진 설치가 성공했는지 확인

```
$ sudo docker run hello-world
```

- 성공



```
ubuntu@ip-172-31-3-53:~$ sudo docker run hello-world

Hello from Docker!
This message shows that your installation appears to be working correctly.

To generate this message, Docker took the following steps:
 1. The Docker client contacted the Docker daemon.
 2. The Docker daemon pulled the "hello-world" image from the Docker Hub.
    (amd64)
 3. The Docker daemon created a new container from that image which runs the
    executable that produces the output you are currently reading.
 4. The Docker daemon streamed that output to the Docker client, which sent it
    to your terminal.

To try something more ambitious, you can run an Ubuntu container with:
$ docker run -it ubuntu bash

Share images, automate workflows, and more with a free Docker ID:
https://hub.docker.com/

For more examples and ideas, visit:
https://docs.docker.com/get-started/
```



## docker-compose 설치

### ✓ docker-compose 사용하기

```
$ sudo apt install docker-compose
```

### ✓ docker-compose.yml 작성

```
vi docker-compose.yml
```

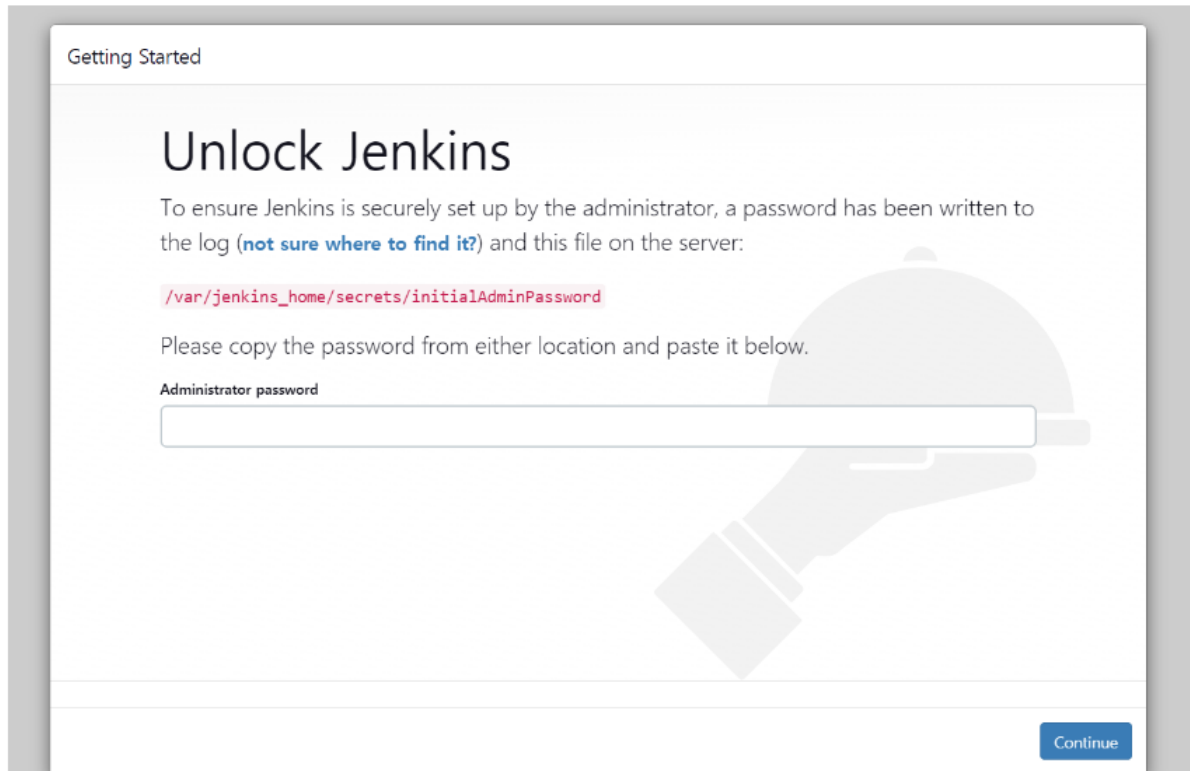
```
version: "3"
services:
  jenkins:
    image: jenkins/jenkins:lts
    container_name: ubuntu_jenkins
    user: root
    volumes:
      - /var/jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 7777:8080
```

- jenkins를 사용
- user : 컨테이너가 실행될 리눅스의 사용자 계정을 root로 명시한다.
- volumes : 도커 컨테이너의 데이터는 컨테이너가 종료되면 휘발된다. 도커 컨테이너의 데이터를 보존하기 위해 사용한다. (/var/jenkins\_home 이라는 디렉터리를 /jenkins와 마운트하고 데이터를 보존할 수 있다.)
- port : 좌측이 호스트 port, 우측이 컨테이너 port (컨테이너 외부와 내부 포트를 포워딩한다.)  
=> docker-compose.yml이 존재하는 경로에서 도커 컨테이너를 실행할 수 있다.

### ✓ 백그라운드로 실행

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

- 성공했다면 k8d103.p.ssafy.io:7777 로 접속해보길 바란다.



## Jenkins 설정

### ✓ Jenkins 초기 password 확인

초기 password는 위 사진의 `Administrator password`에 입력하면 된다.

```
$ sudo docker logs jenkins
```

- jenkins 컨테이너에 출력된 로그를 확인할 수 있다.

또는

```
sudo su
cat /var/jenkins_home/secrets/initialAdminPassword
```

으로 확인할 수 있다.

- 추가적으로 `secrets`는 root 계정만 접근할 수 있다.

### ✓ 젠킨스 접속

비밀번호를 입력하면 아래와 같은 화면이 나온다.

## Customize Jenkins

Plugins extend Jenkins with additional features to support many different needs.

### Install suggested plugins

Install plugins the Jenkins community finds most useful.

### Select plugins to install

Select and install plugins most suitable for your needs.

#### ✓ 젠킨스 plugin 설치

install suggested plugin으로 선택하여 필수적인 plugin들을 설치받도록 한다.

- 혹여나 설치가 실패해도 추후에 검색하여 설치할 수 있으니 실패해도 넘어가면 된다.

#### ✓ 젠킨스 관리자 계정 생성

## Create First Admin User

계정명:	<input type="text" value="jenkins"/>
암호:	<input type="password" value="....."/>
암호 확인:	<input type="password" value="....."/>
이름:	<input type="text" value="jenkins"/>

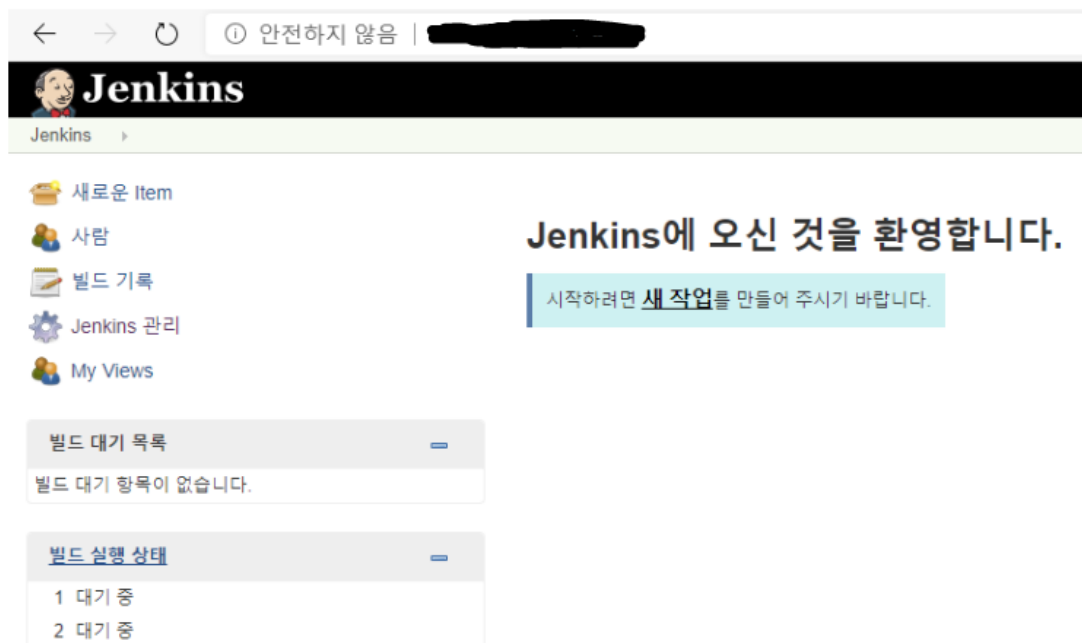
## Getting Started

# Jenkins is ready!

Your Jenkins setup is complete.

Start using Jenkins

### ✓ 젠킨스 접속 완료



### ✓ 타임 존을 서울로 설정하기

```
System.setProperty('org.apache.commons.jelly.tags.fmt.timeZone', 'Asia/Seoul')
```

Jenkins 관리 -> Script console에서 위 스크립트를 입력하면 타임 존이 서울로 설정된다.

### ✓ 플러그인 설치

메뉴 Jenkins 관리 -> 플러그인 관리 -> Available plugins 에서 검색하여 설치

- Docker Pipeline
- NodeJs Plugin
- GitLab

- Generic Webhook Trigger Plugin
- SSH Agent

## ✓ NodeJS 설치

Dashboard > Jenkins 관리 > Global Tool Configuration

### NodeJS

NodeJS installations ^ Edited

NodeJS installations

List of NodeJS installations on this system

Add NodeJS

NodeJS

Name

☒ Install automatically ?

≡ Install from nodejs.org

Version

NodeJS 18.15.0

For the underlying architecture, if available, force the inst

☐ Force 32bit architecture

Global npm packages to install

Specify list of packages to install globally -- see npm ins

Global npm packages refresh hours

Duration, in hours, before 2 npm cache update. Note tha

72

Add Installer ▾

- Name : front
- Version : 18.15.0

## ✓ Credential에 설정(접근 자격 설정)

**ec2\_ssh\_key**: 서버에 접근하기 위한 권한



- 메뉴 Jenkins관리 → Manage Credentials → system → Global credentials
- 우측 상단 Add Credentials 클릭

#### New credentials

Kind

SSH Username with private key

Scope

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

ID

Description

Username

☐ Treat username as secret

Private Key

☒ Enter directly

Key

No Stored Value

Passphrase

Create

- Kind : SSH Username with private key 선택
- ID : 임의의 ID
- Username : ubuntu(키를 받은 우분투 유저 이름 , 보통 Ubuntu)
- Private Key의 Enter directly 체크 후 Add 버튼 클릭
  - server 인스턴스의 pem 파일 내용을 넣어준다.
  - ---BEGIN RSA PRIVATE KEY---부터 ----END RSA PRIVATE KEY----까지 포함해서 다 복사하여 넣어준다.

**gitlab\_token** : GitLab에 접근하기 위한 권한

### New credentials

Kind

Username with password

Scope ?

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

Username ?

☐ Treat username as secret ?

Password ?

ID ?

Description ?

Create

- Kind : Username with password 선택
- ID : 임의의 ID
- Username : GitLab 아이디
- password : GitLab에서 받은 키를 입력한다.
  - 만약 안된다면 GitLab 비밀번호를 넣는 방법도 있다고 한다.)

GitLab에서 key 받기  
User Settings의 Access Tokens 접속

Search page

## 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

Enter the name of your application, and we'll return a unique personal access token.

#### Token name

 gitlab\_token

For example, the application using the token or the purpose of the token. Do not give sensitive information for the name of the token, as it will be visible to all project members.

#### Expiration date

 2023-06-13

#### Select scopes

Scopes set the permission levels granted to the token. [Learn more.](#)

- ☒ **api**  
Grants complete read/write access to the API, including all groups and projects, the container registry, and the package registry.
- ☒ **read\_api**  
Grants read access to the API, including all groups and projects, the container registry, and the package registry.
- ☒ **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 or the Repository Files API.
- ☒ **write\_repository**  
Grants read-write access to repositories on private projects using Git-over-HTTP (not using the API).

Token name : gitlab\_token

Select scopes : 전체 선택

Create personal access token 버튼 클릭하여 생성

생성된 key는 바로 복사해두는 것을 추천한다.

왜냐면 다시 못 보기 때문

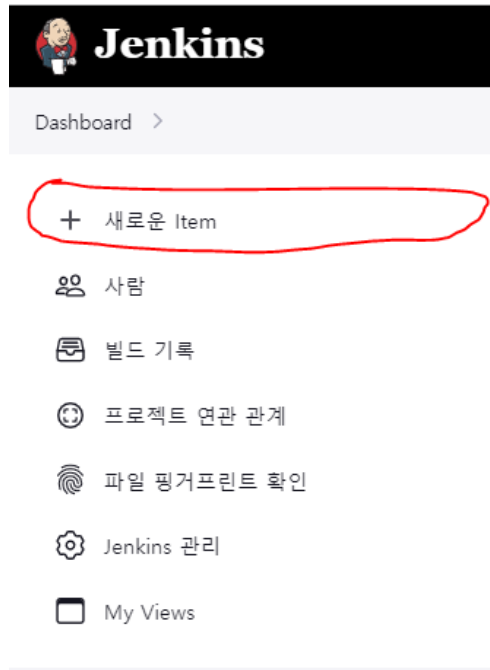
(만약 위의 Username with password가 안된다면

kind : GitLab API token으로 시도해보길 바란다.)

## Pipeline 설정

✓ 새로운 Item 클릭





✓ Pipeline을 선택 후 해당 Pipeline의 이름을 설정하여 생성한다.

 A screenshot of the Jenkins 'Enter an item name' form. At the top is the title 'Enter an item name'. Below it is a text input field. A red error message below the field reads: '> This field cannot be empty, please enter a valid name'. Below the input field are two options: 'Freestyle project' and 'Pipeline'. The 'Pipeline' option is selected, indicated by a red checkmark and a red checkmark icon. The description for 'Pipeline' reads: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly kn type.)'.

✓ 하단의 Definition 을 Pipeline script로 선택

 A screenshot of the Jenkins Pipeline configuration screen. At the top is the title 'Pipeline'. Below it is a dropdown menu labeled 'Definition' with a red checkmark icon. The dropdown menu is open, showing 'Pipeline script' selected. Below the dropdown is a text area labeled 'Script' with a question mark icon. The text area contains a YAML script:
 

```
1 pipeline {
2   agent any
3
4   environment {
5     DOCKER_COMPOSE_VERSION = '1.29.2'
6   }
7
8   stages {
9
```

✓ Back-end pipeline script

```

pipeline {
  agent any

  environment {
    DOCKER_COMPOSE_VERSION = '1.29.2'
  }

  stages {

    stage('gitlab_clone') {
      steps {
        git branch: 'BE', credentialsId: 'gitlab_token', url: 'https://lab.ssafy.com/s08-final/S08P31D103.git'
      }
    }

    stage('secret.yml download') {
      steps {
        withCredentials([file(credentialsId: 'db-credentials', variable: 'dbConfigFile'),
          file(credentialsId: 'login-credentials', variable: 'loginConfigFile'),
          file(credentialsId: 'aws-credentials', variable: 'awsConfigFile')]) {
          script {
            sh 'cp $dbConfigFile Back-end/api-module/src/main/resources/application-db.yml'
            sh 'cp $loginConfigFile Back-end/api-module/src/main/resources/application-login.yml'
            sh 'cp $awsConfigFile Back-end/api-module/src/main/resources/application-aws.yml'

            sh 'cp $dbConfigFile Back-end/batch-module/src/main/resources/application-db.yml'
            sh 'cp $loginConfigFile Back-end/batch-module/src/main/resources/application-login.yml'
            sh 'cp $awsConfigFile Back-end/batch-module/src/main/resources/application-aws.yml'
          }
        }
      }
    }

    stage('build') {
      steps {
        # 'Back-end'폴더 안에서 빌드 실행
        dir('Back-end'){
          sh "chmod +x ./gradlew"
          sh "./gradlew clean build"
        }
      }
    }

    # 이전의 docker container, image 삭제
    stage('docker-clean-up') {
      steps {
        script {
          sshagent(credentials: ['ec2_ssh_key']) {

            sh '''
            if test "`docker ps -aq --filter ancestor=back/api-module`"; then
            # 이전 컨테이너 중지
            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker stop $(docker ps -aq --filter ancestor=back/a
pi-module)"

            # 이전 컨테이너 삭제
            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker rm -f $(docker ps -aq --filter ance
stor=back/api-module)"

            # 이전 이미지 삭제
            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker rmi back/api-module"

            fi
            '''

            sh '''
            if test "`docker ps -aq --filter ancestor=back/batch-module`"; then

            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker stop $(docker ps -aq --filter ancestor=back/b
atch-module)"

            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker rm -f $(docker ps -aq --filter ance
stor=back/batch-module)"

            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker rmi back/batch-module"

            fi
            '''
          }
        }
      }
    }
  }
}

```

```

    }

    stage('docker-build'){
      steps {
        script {
          echo 'Build Docker'
          dir('Back-end') {
            # 'Back-end' 폴더 안에서
            script {
              # docker image를 만들 때 아직 docker가 설치되지 않았다면 설치한다.
              sh """
              if ! test docker; then
                curl -fsSL <https://get.docker.com> -o get-docker.sh
                sh get-docker.sh
              fi
              """
              # 루트 디렉토리에 있는 docker-compose.yml 파일 빌드
              sh 'docker-compose -f docker-compose.yml build'
            }
          }
        }
      }
    }
  }
}

stage('Docker run') {
  steps {
    dir('Back-end') {
      script {
        sh 'docker-compose -f docker-compose.yml up -d'
      }
    }
  }
}
}
}

```

### ✓ 프로젝트 루트 디렉토리

docker-compose.yml

```

version: '3'
services:
  api-module:
    build: ./api-module
    image: back/api-module
    ports:
      - "8081:8081"
  batch-module:
    build: ./batch-module
    image: back/batch-module
    ports:
      - "8082:8082"

```

### ✓ api-module

Dockerfile

```

FROM openjdk:11-jdk

WORKDIR /app

COPY build/libs/api-module-0.0.1-SNAPSHOT-plain.jar api-module.jar

CMD ["java", "-jar", "api-module.jar"]

```

### ✓ batch-module

Dockerfile

```
FROM openjdk:11-jdk

WORKDIR /app

COPY build/libs/batch-module-0.0.1-SNAPSHOT-plain.jar batch-module.jar

CMD ["java", "-jar", "batch-module.jar"]
```

## ✓ Front-end pipeline script

```
pipeline {
  agent any

  tools {nodejs "front"}

  environment {
    DOCKER_COMPOSE_VERSION = '1.29.2'
  }

  stages {
    stage('gitlab_clone') {
      steps {
        git branch: 'FE', credentialsId: 'gitlab_token', url: 'https://lab.ssafy.com/s08-final/S08P31D103.git'
      }
    }

    stage('docker-clean-up') {
      steps {
        script {
          sshagent(credentials: ['ec2_ssh_key']) {

            sh '''
            if test "`docker ps -aq --filter ancestor=front`"; then

            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker stop $(docker ps -aq --filter ancestor=front)"

            // 이전 컨테이너 삭제
            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker rm -f $(docker ps -aq --filter ancestor=front)"

            // 이전 이미지 삭제
            ssh -o StrictHostKeyChecking=no ubuntu@k8d103.p.ssafy.io "docker rmi front"

            fi
            '''
          }
        }
      }
    }

    stage('docker-build'){
      steps {
        script {
          echo 'Build Docker'
          dir('ico') {
            script {

              sh """
              if ! test docker; then
              curl -fsSL <https://get.docker.com> -o get-docker.sh
              sh get-docker.sh
              fi
              """

              sh 'docker-compose -f docker-compose.yml build'
            }
          }
        }
      }
    }
  }
}
```

```

    stage('Docker run') {
        steps {
            dir('ico') {
                script {
                    sh 'docker-compose -f docker-compose.yml up -d'
                }
            }
        }
    }
}
}
}
}

```

## ✓ 프로젝트 루트 디렉토리

docker-compose.yml

```

version: '3'

services:
  app:
    build: .
    image: front
    ports:
      - '3000:3000'
    command: 'npm run start'

```

## Dockerfile

```

# 기본 이미지 설정
FROM node:18.16.0-alpine

# 작업 디렉토리 설정
WORKDIR /app

# 의존성 파일 복사
COPY package.json package-lock.json ./

# 의존성 설치
RUN npm i

# 소스 코드 복사
COPY . .

# 빌드
RUN npm run build

```

## Docker에 MySQL 설치 & 실행

### ✓ 8.0.29 버전으로 설치

```
$ docker pull mysql:8.0.29
```

### ✓ 3306포트로 실행

```
$ docker run -d -p 3306:3306 -e MYSQL_ROOT_PASSWORD=<비밀번호> --name <도커 컨테이너 이름> mysql:8.0.29 --character-set-server=utf8mb4 --collation-server=utf8mb4_unicode_ci
```

## MySQL bash 접속, 계정 생성

## ✓ bash 접속

```
$ docker exec -it <컨테이너 이름> bash
```

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

## ✓ 데이터베이스 목록 확인

```
mysql> show databases;
```

```
mysql> show databases;
+-----+
| Database |
+-----+
| information_schema |
| mysql      |
| performance_schema |
| sys        |
| themint_db |
+-----+
5 rows in set (0.18 sec)
```

## ✓ mysql 데이터베이스로 접속

```
mysql> use mysql;
```

## ✓ 유저 생성

```
mysql> create user '<계정이름>'@'%' identified by '<비밀번호>';
```

\* 계정 생성과 비밀번호 설정이 동시에 안 된다면 (sql syntax error)

`identified by '<비밀번호>'` 부분은 떼고 실행한 다음에

```
mysql> alter user '<계정이름>'@'%' identified by '<비밀번호>';
```

## ✓ 유저에 권한 부여

```
mysql> grant all privileges on *.* to '아이디'@'localhost';
```

```
mysql> FLUSH PRIVILEGES;
```

## Nginx 설치

### ✓ Nginx 설치와 버전 확인

```
sudo apt-get install nginx  
  
nginx -v
```

### ✓ letsencrypt 인증서 발급

```
sudo apt-get install nginx  
nginx -v
```

이걸 실행했을 때 Congratulations! 이 보이면 성공한거고

시간이 뜨면서 기다려 달라는 메시지를 띄우면 그 시간 까지 기다렸다 발급받으면 된다.

`/etc/nginx/sites-available` 로 이동한 이후

`sudo vi proxy-setting` 파일을 하나 만들고

```
server {  
    listen 80 default_server;  
    listen [::]:80 default_server;  
    server_name k8d103.p.ssafy.io;  
  
    location /{  
        proxy_pass http://localhost:3000;  
        proxy_set_header Connection '';  
        proxy_http_version 1.1;  
    }  
  
    location /api {  
        proxy_pass http://localhost:8081/api;  
    }  
}  
  
server {  
    listen 443 ssl;  
    listen [::]:443 ssl;  
  
    server_name k8d103.p.ssafy.io;  
  
    location /{  
        proxy_pass http://localhost:3000;  
        proxy_set_header Connection '';  
        proxy_http_version 1.1;  
    }  
  
    location /api {  
        proxy_pass http://localhost:8081/api;  
    }  
  
    ssl_certificate /etc/letsencrypt/live/k8d103.p.ssafy.io/fullchain.pem; # managed by Certbot  
    ssl_certificate_key /etc/letsencrypt/live/k8d103.p.ssafy.io/privkey.pem; # managed by Certbot  
}
```

80은 http, 443은 https 로 접근하는 것을 의미한다.

- SSE 지속 연결 설정

```
proxy_set_header Connection '';
proxy_http_version 1.1;
```

Nginx 기본 포트가 80로 실행되기 때문에

`/etc/nginx/sites-enabled` 에서

```
sudo vi default
```

80 → 안쓰는 포트 번호(예시 180)으로 바꾼다.

```
server {
    listen 180 default_server;
    listen [::]:180 default_server;
```

✓ `ln -s` 명령어 실행한다.

```
sudo ln -s /etc/nginx/sites-available/proxy-setting /etc/nginx/sites-enabled/proxy-setting
```

✓ 성공 여부 확인

```
sudo nginx -t
```

```
sudo systemctl restart nginx
```

✓ `nginx.conf`

`etc/nginx/` 에서

```
vi nginx.conf
```

업로드 가능한 파일 최대 크기 확장 옵션 추가

```
http {
    ##
    # Basic Settings
    ##
    client_max_body_size 100M;
    sendfile on;
    # ...
```

테스트 확인

```
sudo nginx -t
```



nginx를 재 시작해서 변경사항을 적용

```
nginx -s reload
```