



포팅 메뉴얼

🕒 생성일	@2023년 10월 2일 오후 11:08
☰ 태그	

1. 개발환경

- [1.1. Frontend](#)
- [1.2 Backend](#)
- [1.3 Galaxy Watch](#)
- [1.4 Server](#)
- [1.5 Database](#)
- [1.6 UI/UX](#)
- [1.7 IDE](#)
- [1.8 형상 / 이슈관리](#)
- [1.9 기타 툴](#)

2. EC2 세팅

- [2.1 Docker 설치](#)
- [2.2 Nginx 설치](#)
- [2.3 SSL 적용](#)
- [2.4 EC2 Port](#)
 - [EC2 1 \(Front, API-Server\)](#)
- [2.4 방화벽 설정](#)

4. CI/CD 구축

- [4.1. Jenkins 도커 이미지 + 컨테이너 생성](#)
- [4.2. Jenkins 설정](#)
 - [4.2.1 GitLab Credentials 설정](#)
 - [Stores scoped to Jenkins](#)
 - [4.2.1 Jenkins item 생성](#)
 - [4.2.3. GitLab Webhook 설정](#)

5. 외부서비스

- [5.1. 소셜로그인 - Kakao](#)
 - [카카오 디벨로퍼](#)
 - [1. 사이트 도메인 등록](#)
 - [2. 안드로이드 등록](#)
 - [3. 카카오 로그인 활성화](#)
 - [4. 리다이렉트 URI 설정](#)
- [5.2. 소셜로그인 - Naver](#)
 - [네이버 개발자](#)
 - [1. 리다이렉트 URI 지정](#)
 - [2. 안드로이드 URL 지정](#)

yaml

요청 흐름

6. 갤럭시 위치

- [1. 갤럭시 위치 설정](#)
- [2. 안드로이드 스튜디오 설정](#)

1. 개발환경

1.1. Frontend

- React (Web)
- React-native (App)
 - Recoil
- Typescript
- React-native styled-components

1.2 Backend

- Java
 - Java OpenJDK 11
 - Spring Book 2.7.15
 - Spring Data JPA
 - Spring Security
 - Lombok
 - JWT
 - Oauth 2.0
 - Gradle {version}

1.3 Galaxy Watch

- WearOS 4
- Retrofit 2 : 2.9.0
- Room : 2.5.0
- GSON : 2.10.1
- Brotli : 0.1.2

1.4 Server

- Ubuntu
- Nginx
- Docker
- Docker Compose
- Jenkins

1.5 Database

- MySQL
- Redis
- Amazon S3
- MongoDB

1.6 UI/UX

- Blender, Figma, Canva,

1.7 IDE

- Visual Studio Code
- IntelliJ IDEA
- Android Studio

1.8 형상 / 이슈관리

- Gitlab
- Jira

1.9 기타 툴

- Postman
- Mattermost
- Notion

2. EC2 세팅

2.1 Docker 설치

```
### install-docker.sh

# update apt packages
sudo apt-get update

# install docker engine
sudo apt-get install -y docker docker-io

# test
docker ps
```

2.2 Nginx 설치

```
# 1. Nginx 설치
sudo apt-get install nginx

# 2. start
sudo systemctl status nginx
sudo systemctl start nginx

nginx -v
# 2. Let's Encrypt 설치 및 SSL 발급
sudo apt-get install letsencrypt
sudo systemctl stop nginx
sudo letsencrypt certonly --standalone -d 도메인명

# 3. Nginx 설정파일 생성
cd /etc/nginx/sites-available
vi configure
```

2.3 SSL 적용

```
sudo apt-get update

# letsencrypt 설치
sudo apt-get install letsencrypt

# ssl 인증서 생성
sudo certbot certonly --standalone

# 인증서 생성하면서, 이메일과, EC2 도메인 입력!

# 인증서 생성 후, /etc/letsencrypt/live/{도메인}에 인증서 파일 확인!
```

```
# /etc/nginx/site-available/default 파일 수정

##
# You should look at the following URL's in order to grasp a solid understanding
# of Nginx configuration files in order to fully unleash the power of Nginx.
# https://www.nginx.com/resources/wiki/start/
# https://www.nginx.com/resources/wiki/start/topics/tutorials/config_pitfalls/
# https://wiki.debian.org/Nginx/DirectoryStructure
#
# In most cases, administrators will remove this file from sites-enabled/ and
# leave it as reference inside of sites-available where it will continue to be
# updated by the nginx packaging team.
#
# This file will automatically load configuration files provided by other
```

```

# applications, such as Drupal or Wordpress. These applications will be made
# available underneath a path with that package name, such as /drupal8.
#
# Please see /usr/share/doc/nginx-doc/examples/ for more detailed examples.
##

# Default server configuration
#

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

    server_name dallim.site;
    server_tokens off;

    access_log /var/log/nginx/reverse-access.log;
    error_log /var/log/nginx/reverse-error.log;

    location / {
        return 301 https://$host$request_uri;
    }
}

server {
    listen 443 ssl;
    server_name dallim.site;
    server_tokens off;

    ssl_certificate /etc/letsencrypt/live/dallim.site/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/dallim.site/privkey.pem;

    brotli_comp_level 2;
    brotli_types application/json;
    brotli_static on;
    brotli_buffers 16 8k;

    location / {
        proxy_pass http://localhost:3001;
    }
    location /api {
        proxy_pass http://localhost:8081;
    }
    location /brotli {
        brotli on;
        proxy_pass http://localhost:8081;
    }
    location /swagger {
return 301 http://k9b208.p.ssafy.io:8081/swagger-ui/index.html;
    }

    location /login {
        proxy_pass http://localhost:8081;
    }

    location /oauth2 {
        proxy_pass http://localhost:8081;
    }

    location /grafana {
        return 301 http://k9b208.p.ssafy.io:3000;
    }

    location /prometheus {
        return 301 http://k9b208.p.ssafy.io:9091;
    }

    location /actuator {
        proxy_pass http://localhost:8081;
    }
}

```

2.4 EC2 Port

EC2 1 (Front, API-Server)

Port 번호	내용
22	SSH

Port 번호	내용
80	HTTP(HTTPS로 redirect)
443	HTTPS
3001	Nginx, React (Docker)
6379	Redis (JWT Token)
8080	Spring Boot (Docker)
32773	Jenkins
9090	Prometheus
3000	Grafana
8086	influxDB
27017	mongoDB
3306	mySQL

2.4 방화벽 설정

```
# EC2 1 (Front, API-Server) 방화벽 확인
sudo ufw status

# 1. 해당 포트 개방
# 22 TCP
# 80 TCP

sudo ufw allow 22
sudo ufw allow 80

# Firewall 활성화 상태 확인
sudo ufw enable
sudo ufw status verbose

# 3. Nginx reverse proxy 설정 후 Frontend, Backend, Jenkins 서버 포트 닫기
sudo ufw deny 3000/tcp # React Nginx
sudo ufw deny 8080/tcp # Spring Boot
sudo ufw deny 32773/tcp # Jenkins

# EC2 2 방화벽 확인
sudo ufw status

# 1. 해당 포트 개방
# 22 TCP
# 80 TCP

sudo ufw allow 22
sudo ufw allow 80

# Firewall 활성화 상태 확인
sudo ufw enable
sudo ufw status verbose

# 3. Nginx reverse proxy 설정 후 API Server, Jenkins 서버 포트 닫기
sudo ufw deny 8080/tcp # Spring Server
sudo ufw deny 32773/tcp # Jenkins
```

4. CI/CD 구축

4.1. Jenkins 도커 이미지 + 컨테이너 생성

```
# jenkins 실행
sudo docker run --name jenkins -d -p 32773:8080 -p 50000:50000 -v /var/jenkins_home:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -u root jenkins/jenkins

# jenkins 접속
sudo docker exec -it jenkins bash

# jenkins에 docker 설치
apt-get update
apt-get install docker docker.io
```

```
# docker 확인
docker ps
```

4.2. Jenkins 설정

[CI/CD] Gitlab과 Jenkins로 CI/CD 구축하기

Gitlab과 Jenkins로 CI/CD 구축하기

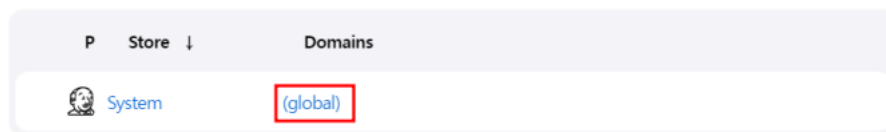
 <https://velog.io/@haniff/Gitlab과-Jenkins로-CICD-구축하기>

ve1og

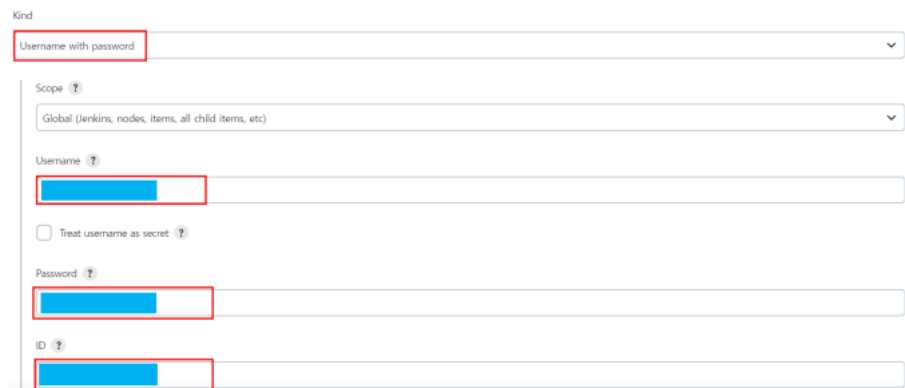
4.2.1 GitLab Credentials 설정

1. jenkins 관리 → “Credentials” 클릭
2. “Store : System” → “(global)” → “+ Add Credentials” 클릭

Stores scoped to Jenkins



3. “Username with password” 입력 → “Username” 에 GitLab ID 입력 → “password에 Gitlab Personal Access Tokens 입력” → “ID”에 임의 아이디 입력 → 생성



4.2.1 Jenkins item 생성

1. item 생성
2. “Enter an item name” 에 임의 item 이름 입력 → Pipeline 클릭



3. "Build Triggers" → "Build when a change is pushed to GitLab" 클릭 (WebHook 설정: GitLab 특정 브랜치 push 시 자동 빌드 → 배포 설정)

☒ Build when a change is pushed to GitLab. GitLab webhook URL: [Redacted] ?

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 ▼

4. pipeline 작성

Pipeline

Definition

Pipeline script ▼

Script ?

```

18 stages {
19   stage('Delete Repository'){
20     steps{
21       sh '''
22         rm -rf /var/jenkins_home/workspace/${workspaceName}/S09P31B208 || true
23       '''
24     }
25   }
26
27   stage('Checkout') {
28     steps {
29       withCredentials([string(credentialsId: 'gitlab-token', variable: 'ACCESS_TOKEN')]) {
30         // GitLab 레포지토리를 클론하는 단계
31         sh '''
32         git clone https://gitlab.com/seongli/dallim-backend.git --depth 1 --branch S09P31B208 --single-branch
33       '''
34     }
35   }
36 }

```

☒ Use Groovy Sandbox ?

```

pipeline {
    environment{
        repository = "seongli/dallim-backend"
        DOCKERHUB_CREDENTIALS = credentials('dockerHub')
        containerName = "${repository}"
        containerPort = 8080
        imageName = "${containerName}:latest"
        workspaceName = "backend-pipeline"
        IDLE_PORT = -1
        SELECTED_PORT = -1
    }

    agent any

    stages {
        stage('Delete Repository'){
            steps{
                sh '''
                rm -rf /var/jenkins_home/workspace/${workspaceName}/S09P31B208 || true
                '''
            }
        }
    }
}

```

```

    }
}

stage('Checkout') {
    steps {
        withCredentials([string(credentialsId: 'gitlab-token', variable: 'ACCESS_TOKEN')]) {
            // GitLab 레포지토리를 클론하는 단계
            sh '''
                git clone -b backend https://gitlab-ci-token:${ACCESS_TOKEN}@lab.ssafy.com/s09-final/S09P31B208.git

                mkdir -p /var/jenkins_home/workspace/${workspaceName}/S09P31B208/backend/dduishu/src/main/resources/
                mkdir -p /var/jenkins_home/workspace/${workspaceName}/S09P31B208/backend/dduishu/src/main/resources/firebase
                cp /var/jenkins_home/workspace/secret/application.yml /var/jenkins_home/workspace/${workspaceName}/S09P31B208/back
                cp /var/jenkins_home/workspace/secret/application-connection.yml /var/jenkins_home/workspace/${workspaceName}/S09
                cp /var/jenkins_home/workspace/secret/application-oauth2.yml /var/jenkins_home/workspace/${workspaceName}/S09P31B
                cp /var/jenkins_home/workspace/secret/firebase_service_key.json /var/jenkins_home/workspace/${workspaceName}/S09P

                cp /var/jenkins_home/workspace/secret/logback.xml /var/jenkins_home/workspace/${workspaceName}/S09P31B208/backend
                cp /var/jenkins_home/workspace/secret/file-appender.xml /var/jenkins_home/workspace/${workspaceName}/S09P31B208/1
                cp /var/jenkins_home/workspace/secret/console-appender.xml /var/jenkins_home/workspace/${workspaceName}/S09P31B20

            '''
            sh 'pwd'
        }
    }
}

stage('Build Jar'){
    tools {
        // Gradle을 사용할 수 있도록 설정합니다.
        //위에서 설정한 Gradle 이름
        gradle 'Gradle'
    }
    steps {
        // Gradle을 실행하여 Jar 빌드를 수행합니다.
        //프로젝트에서 gradlew가 있는 위치로 이동
        sh '''
            cd /var/jenkins_home/workspace/${workspaceName}/S09P31B208/backend/dduishu
            ls -al
            chmod +x gradlew
            ./gradlew build
            '''
    }
}

// 테스트 (예제는 Java 프로젝트를 대상으로 함)
stage('Test') {
    steps {
        echo "Testing.."
        sh '''
            echo "doing test stuff.."
            '''
    }
}

stage('Building our image'){
    steps{
        //build한 프로젝트 jar을 이미지로 만들기 위해 test프로젝트에서 dockerhub 프로젝트로 이동
        //build를 위한 Dockerfile은 dockerhub 프로젝트의 루트 디렉토리에 있어야 한다.
        script{
            sh '''
                pwd
                cd /var/jenkins_home/workspace/${workspaceName}/S09P31B208/backend/dduishu
                cp /var/jenkins_home/workspace/${workspaceName}/S09P31B208/backend/dduishu/build/libs/dduishu-0.0.1-SNAPSHOT.jar /

                docker build -t ${repository}:${BUILD_NUMBER} .
                docker build -t ${repository}:latest .
            '''
        }
    }
}

stage('Login'){
    steps{
        sh 'echo $DOCKERHUB_CREDENTIALS_PSW | docker login -u $DOCKERHUB_CREDENTIALS_USR --password-stdin'
    }
}

stage('Deploy Hub our image'){
    steps{
        script{
            sh 'docker push $containerName:$BUILD_NUMBER'
            sh 'docker push $containerName:latest'
        }
    }
}

stage('Stop Container'){
    steps{
        script{
            sh '''

```



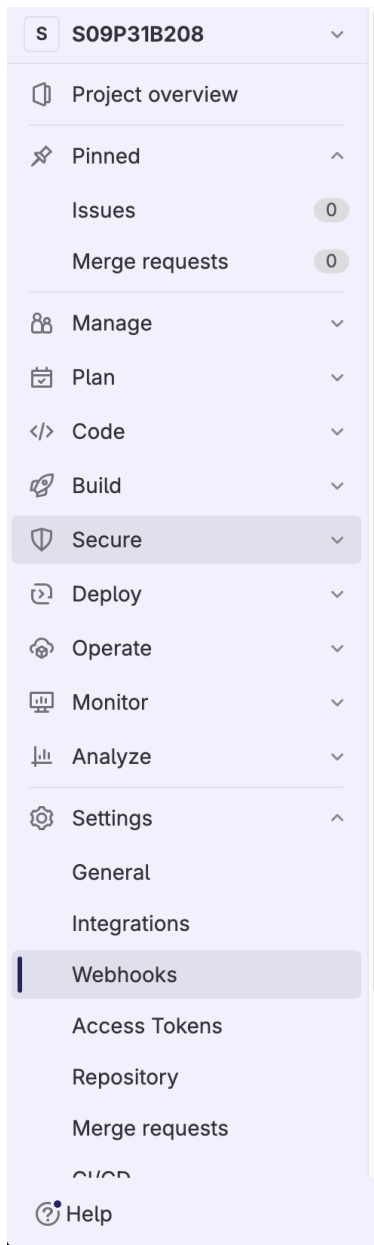
```

        pwd
        docker stop API-Server || true
        docker rm API-Server || true
        docker rmi $containerName:latest || true
        pwd
    ''
}
}
}
stage('Deploy Container'){
    steps{
        script{
            sh "docker run -v /home/ubuntu/log:/app/log --network promnet -d -p 8081:8081 --name API-Server ${imageName}"
        }
    }
}
stage('Cleaning up'){
    steps{
        sh ''
        rm -rf /var/jenkins_home/workspace/${workspaceName}/S09P31B208
        rm -rf /var/jenkins_home/workspace/${workspaceName}/dduishu-0.0.1-SNAPSHOT.jar
        ''
    }
}
}
}
}

```

4.2.3. GitLab Webhook 설정

1. 프로젝트 GitLab → setting → webhook 클릭



2. jenkins URL 입력 → Secret token 입력 → push events 클릭

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

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.

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

Wildcards such as `*-stable` or `production/*` are supported.

☐ Regular expression

5. 외부서비스

5.1. 소셜로그인 - Kakao

카카오 디벨로퍼

1. 사이트 도메인 등록

Web		삭제	수정
사이트 도메인	https://dallim.site		
• 카카오 로그인 사용 시 Redirect URI를 등록해야 합니다. 등록하러 가기			

2. 안드로이드 등록

Android		삭제	수정
패키지명	com.dallim		
마켓 URL	market://details?id=com.dallim		
키 해시	alsoskbV2VdbOxrBLPOCVmNfZos=		

3. 카카오 로그인 활성화

카카오 로그인

ON

[동의 화면 미리보기](#)

활성화 설정

상태

ON

카카오 로그인 API를 활용하면 사용자들이 번거로운 회원 가입 절차 대신, 카카오톡으로 서비스를 시작할 수 있습니다.
상태가 OFF일 때도 카카오 로그인 설정 항목을 변경하고 서버에 저장할 수 있습니다.
상태가 ON일 때만 실제 서비스에서 카카오 로그인 화면이 연결됩니다.

4. 리다이렉트 URI 설정

Redirect URI		삭제	수정
Redirect URI	http://localhost:8080/login/oauth2/code/kakao https://k9b208.p.ssafy.io/login/oauth2/code/kakao		
• 카카오 로그인에서 사용할 OAuth Redirect URI를 설정합니다. (최대 10개)			
• REST API로 개발하는 경우 필수로 설정해야 합니다.			

웹뷰를 통해 Frontend에서 인가 code인 `code?{authcode}` 이 authcode를 가져오고, 가져온 authcode를 이용하여 백으로 요청을 보내고, 백에서 이 인가 코드를 통해 kakao에게 요청을 진행함.

5.2. 소셜로그인 - Naver

네이버 개발자

1. 리다이렉트 URI 지정

로그인 오픈 API
서비스 환경 ①

네이버 로그인
Callback URL (최대 5개)

http://localhost:8080/login/oauth2/code/naver

-

https://dallim.site/login/oauth2/code/naver

+ ✓

텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다.
Callback URL은 네이버 로그인 후 이동할 페이지 URL입니다. Callback URL 값이 잘못 입력되어 있으면 정확한 값으로 수정하실 때 까지 네이버 로그인 사용이 일시적으로 제한됩니다.
입력한 주소와 다른 Callback URL로 리다이렉트 될 경우, 이용이 제한될 수 있습니다.

안드로이드 ✕ ^

다운로드 URL

https://play.google.com/store/apps/details?id=com.dallim

불법/음란성 사이트 등 이용약관에 위배되는 사이트의 경우, 이용이 제한될 수 있습니다.

네이버 소셜 로그인은 검수 요청을 받아야 외부 인원이 로그인 서비스를 이용할 수 있다.

검수를 받지 않으면 테스트 계정 ID를 등록해놓은 유저만 로그인 기능이 이용 가능함. (즉 검수가 필수)

2. 안드로이드 URL 지정

안드로이드 ✕ ^

다운로드 URL

https://play.google.com/store/apps/details?id=com.dallim

불법/음란성 사이트 등 이용약관에 위배되는 사이트의 경우, 이용이 제한될 수 있습니다.

안드로이드 앱 패키지 이름

com.dallim

+

텍스트 폼 우측 끝의 '+' 버튼을 누르면 행이 추가되며, '-' 버튼을 누르면 행이 삭제됩니다.
2015년 8월 7일 이전에 등록된 앱은 앱 패키지 이름 대신 안드로이드 Intent가 등록되어 있어도 동작합니다.

yaml

```

spring:
  security:
    oauth2:
      client:
        registration:
          naver:
            clientId: {네이버id}
            clientSecret: {네이버 시크릿키}
            clientAuthenticationMethod: post
            authorizationGrantType: authorization_code
            redirectUri: http://{서버 주소}/login/oauth2/code/naver
            scope: nickname, email, profile_image
            clientName: Naver
          kakao:
            client-id: {카카오 id}
            client-secret: {카카오 시크릿키}
            scope:
              - account_email
              - profile_nickname
              - profile_image
            authorization-grant-type: authorization_code
            redirect-uri: http://{서버주소}/login/oauth2/code/kakao
            client-name: Kakao
            client-authentication-method: POST
        provider:
          naver:
            authorizationUri: https://nid.naver.com/oauth2.0/authorize
            tokenUri: https://nid.naver.com/oauth2.0/token
            userInfoUri: https://openapi.naver.com/v1/nid/me
            userNameAttribute: response
          kakao:
            authorizationUri: https://kauth.kakao.com/oauth/authorize
            tokenUri: https://kauth.kakao.com/oauth/token
            userInfoUri: https://kapi.kakao.com/v2/user/me
            userNameAttribute: id

```

요청 흐름

1. 프론트엔드 로그인 요청 주소
2. `https://nid.naver.com/oauth2.0/authorize?client_id={네이버 클라이언트 아이디}&response_type=code&redirect_uri={리다이렉트 uri}` 로 로그인 요청을 보냄
3. 프론트엔드에서 리다이렉트 uri로 넘어온 `code={authCode}` 를 통해서 백엔드로 요청을 보냄
4. 백엔드에서 넘어온 `authCode`를 이용하여 카카오에 정보를 요청하여 설정해 놓은 정보들(ex. 닉네임, 사진, 아이디)을 받아옴.
5. 정보들을 통해 db에 해당 유저가 없으면 유저를 생성해주고, 회원가입을 진행함.
6. 회원가입 진행이 완료가 되면 프론트엔드로 `accessToken`을 보내준다.
7. `accessToken`을 갖고 redis에 저장한 정보와 일치하는지 확인하여 우리 서비스의 로그인을 진행하게 된다.

6. 갤럭시 위치

안드로이드 스튜디오와 갤럭시 위치를 WIFI 연결해서 실제 기기 테스트 하는 방법을 소개합니다.

사용하는 PC와 갤럭시 위치가 같은 WIFI를 사용하고 있어야 가능합니다.

1. 갤럭시 위치 설정

설정 → 위치 정보 → 소프트웨어 정보 → 소프트웨어 버전 광클하면 개발자 모드가 켜집니다.

개발자 옵션 → ADB 디버깅 체크 → 무선 디버깅을 켜줍니다. 이후 새 기기 등록 선택

2. 안드로이드 스튜디오 설정

File → Project Structure → SDK Location 들어가서 경로를 복사해줍니다.

터미널로 이동해서 `cd {복사 경로}\platform-tools` 를 입력해서 해당 디렉토리로 이동해줍니다.

`./adb pair {ip주소}:{포트번호} {페어링 코드}` 를 입력해서 페어링을 먼저 진행해줍니다.

```
ex) ./adb pair 123.123.123.123:12345 123456
```

이후 위치 화면에 보이는 ip와 포트를 연결해줍니다.

```
ex ./adb connect 123.123.123.123:12345
```

연결이 성공했다고 뜨면 안드로이드 스튜디오 위에 갤럭시 위치 모델명이 연결된 것을 확인할 수 있습니다.

