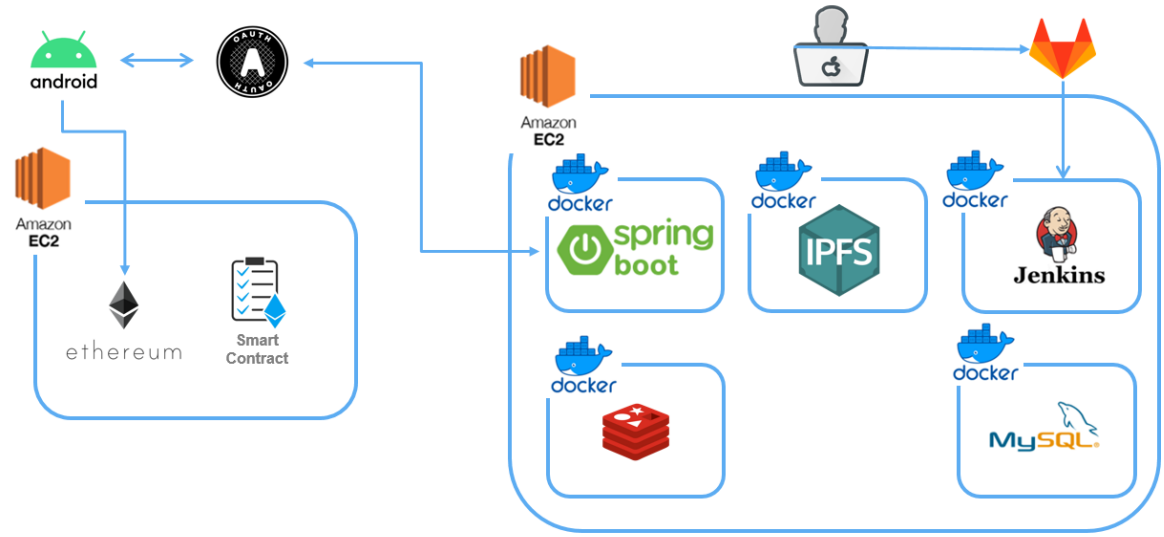
포팅 매뉴얼

1. 프로젝트 기술 스택

AOS	• Retrofit 2.9.0 (통신 라이브러리) • OkHttp • Ted Permission 3.3.0 (안드로이드 권한 라이브러리) • Dagger-Hilt (의존성 주입 라이브러리) • JetPack Paging3 (페이징 라이브러리) • Coroutines Flow (비동기 데이터 처리 라이브러리) • Glide 4.12.0 (이미지 로드 라이브러리) • ViewModel-ktx 2.3.1 • Fragment-ktx 1.3.6 • Navigation 2.3.5 (화면 전환, 스택 관리 라이브러리) • OAuth (로그인 보안 라이브러리) • ExoPlayer (미디어 플레이어 라이브러리) • Room (내부 데이터 베이스 라이브러리) • Biometric (생체 인식 라이브러리) • Zxing Qr Scan (QR 스캔 라이브러리)
BE	**Infra** • AWS EC2 • Docker 20.10.17 • Docker-compose 1.25.0 • Jenkins 2.346.2 **Development** • Java 1.8.0_192(Zulu 8.33.0.1-win64) • Spring boot 2.7.3 • spring-data-jpa 2.7.3 • spring-data-redis 2.7.3 • hibernate-core-5.6.9.Final • spring-security:5.7.3 • projectlombok:1.18.24 **Test** • junit-jupiter:5.8.2 • mockito-core:4.5.1 • Apache JMeter 5.5 **DB** • mysql 8.0.28
Blockchain	**Infra** • AWS EC2 **Language** • Solidity 0.8.0 • Go **Package** • Web3j 4.8.7 • Geth 1.10.25-stable • Nodejs v12.22.9 • npm 8.5.1 • ethereumjs-wallet 0.6.5

2. 서버 아키텍처



본 프로젝트의 아키텍처는 위와 같습니다. 각 서버 리소스는 특정 포트로 식별 가능하며 접근할 수 있습니다.

각 서버의 포트 번호는 다음과 같습니다. (도커 배포 기준)

서버	HTTP 포트	HTTPS 포트
tomcat	8080	8443
jenkins	8088	-
redis	6379	-
mysql	3306	-

3. 프로젝트 빌드 방법 (로컬 서버)

3.1. Gitlab에서 프로젝트 클론하기

1. 작업할 공간에 폴더를 하나 생성합니다.
2. 생성한 폴더를 열고 해당 위치에서 `Git Bash`를 열어줍니다. (`CMD`와 같은 다른 터미널도 상관없습니다!)
3. `git clone https://lab.ssafy.com/s07-blockchain-contract-sub2/S07P22D102.git`를 터미널에 입력해줍니다.
4. gitlab에서 내려받은 파일이 생깁니다. 앞으로 해당 파일이 위치한 폴더를 `root directory`라고 하겠습니다. 이후 작업 공간으로 가서 빌드 과정을 수행해주시면 됩니다.

3.2. 스프링부트 WAS 빌드

3.2.1. gradle로 직접 빌드하는 방법 (CMD 버전)

1. `Win + R`을 누르고 `cmd`를 입력하고 확인 버튼을 누릅니다. 그러면 명령 프롬프트 창을 띄울 수 있습니다.
2. 백엔드 작업 공간으로 이동해줍니다. 저의 경우에는 백엔드 작업 공간이 `C:\Users\multicampus\Desktop\specialization-project\BE\indive`입니다. 앞에 `cd` 명령어를 붙이시면 해당 디렉토리로 이동할 수 있습니다.
3. 그 후 `gradle`을 이용하여 빌드해줍니다. `cmd`에 `gradlew clean build` 명령어를 입력합니다. 그러면 서버 내부에서 진행하는 테스트 코드를 수행한 후 빌드 파일이 생깁니다.

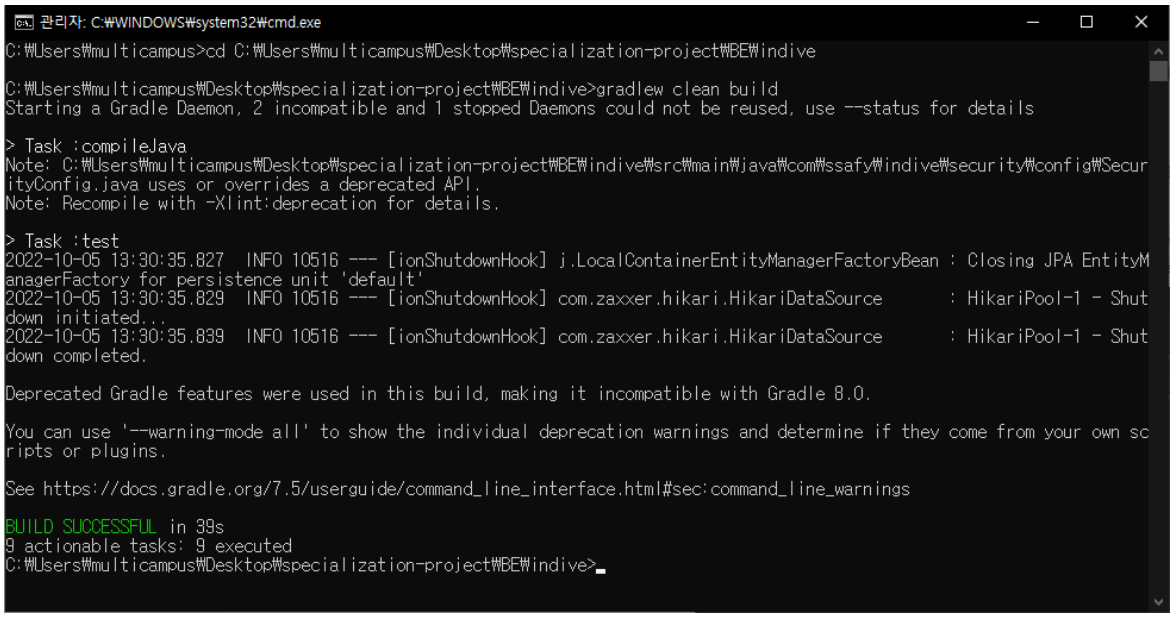


그림 2) gradle build

4. `cd build/libs` 명령어를 입력해서 빌드 파일이 있는 위치로 이동한 후 `java -jar indive-0.0.1-SNAPSHOT.jar` 명령어를 입력해줍니다.
5. 위의 과정을 마치면 로컬 환경에서 서버 빌드 및 배포가 되었습니다. `http://localhost:8080/swagger-ui/`로 접속하시면 API를 확인하실 수 있습니다.

3.2.2. gradle로 직접 빌드하는 방법 (IntelliJ)

1. 인텔리제이를 통해 해당 프로젝트를 열어줍니다.

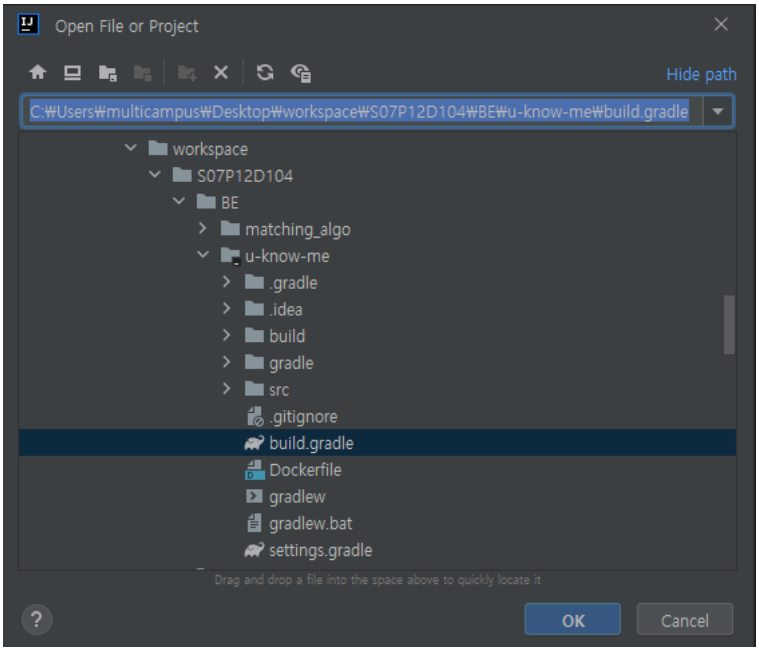


그림 3) 오픈 프로젝트를 통해 백엔드 프로젝트 열기

2. `Alt + F12`를 눌러 터미널을 열어줍니다.

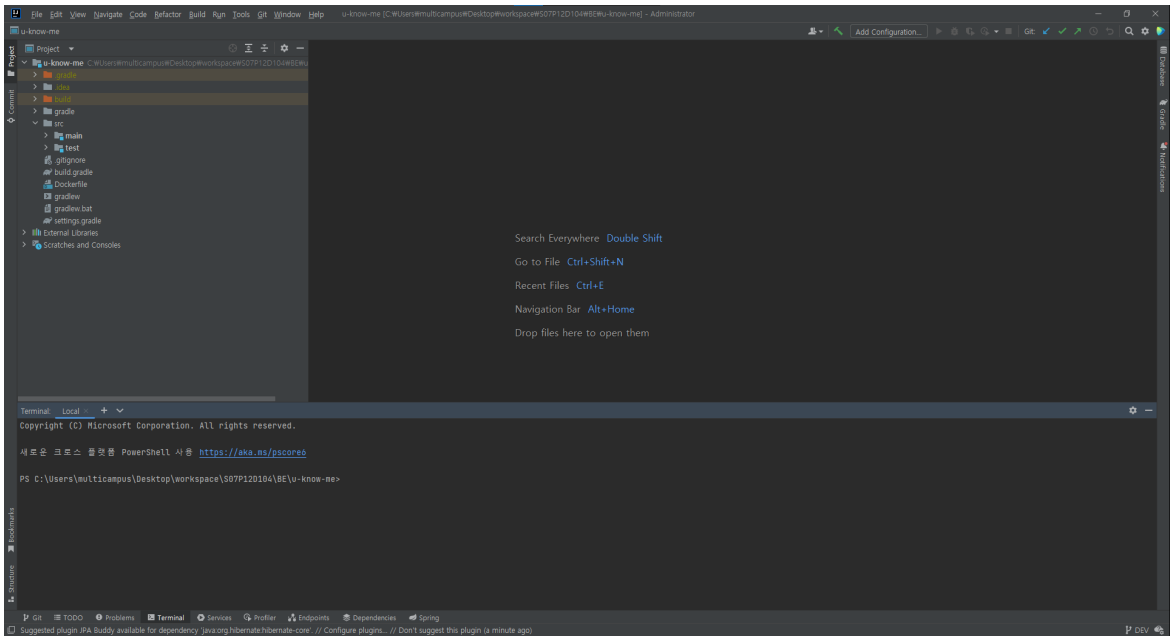


그림 4) 터미널을 연 상태의 IntelliJ

3. 터미널에 `./gradlew clean build` 명령어를 입력합니다.
4. `BUILD SUCCESSFUL`이 뜨면 `cd build/libs` 명령어를 입력해서 빌드 파일이 있는 위치로 이동한 후 `java -jar indive-0.0.1-SNAPSHOT.jar` 명령어를 입력해줍니다.
5. 스프링부트 서버가 정상적으로 올라가면 `http://localhost:8080/swagger-ui/`로 접속하셔서 API를 확인하실 수 있습니다.

4. 프로젝트 빌드 방법 (운영 서버)

4.1. VSCode를 이용한 ssh 접속

1. `Remote - SSH` **설치**

vscode에서 `ctrl + shift + x`를 누르면 extensions 탭으로 넘어갈 수 있습니다. 해당 화면에서 검색창에 `ssh`를 검색하면 `Remote - SSH` 라는 extension이 나오는데 `install` 버튼을 눌러 다운로드 받아줍니다.

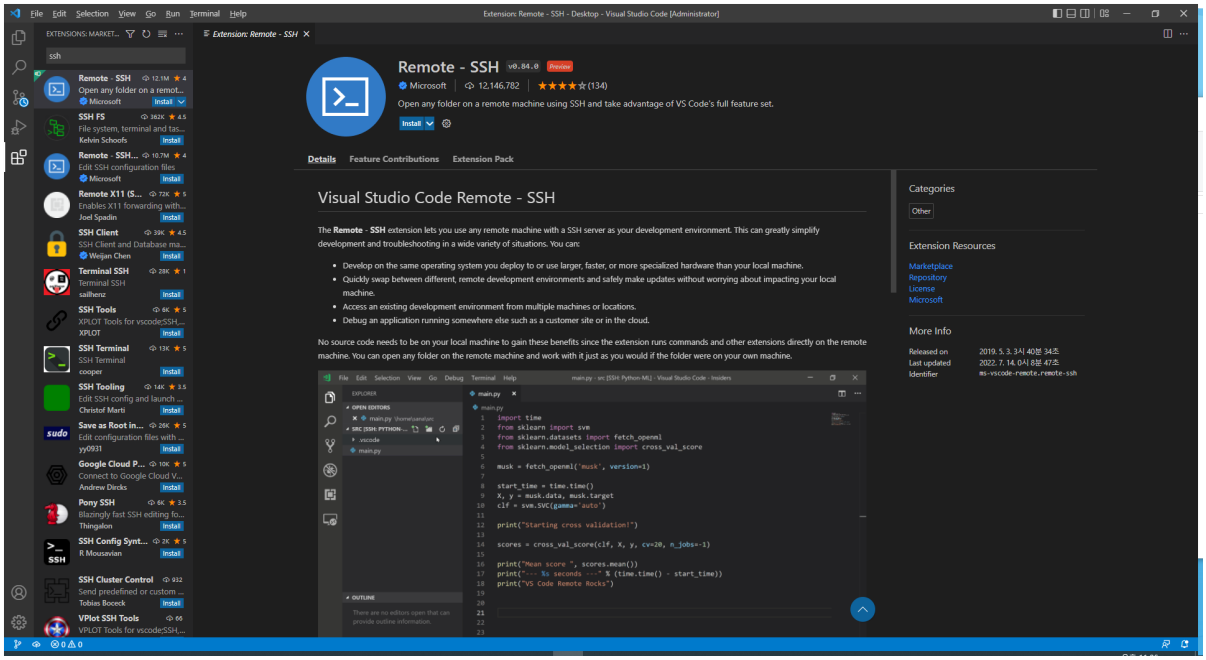


그림 5) extensions 탭

2. **SSH 설정 파일 등록**

원래 터미널에서 `ssh -i 계정명@IP주소`로 연결할 수 있지만 계속 터미널에 명령어를 입력하기는 번거로우니 설정 파일을 등록해줍니다.

우선 `f1` 버튼을 눌러 `ssh`를 검색합니다. 그리고 `Remote-SSH:Open SSH Configuration File...`이라는 탭을 선택합니다.

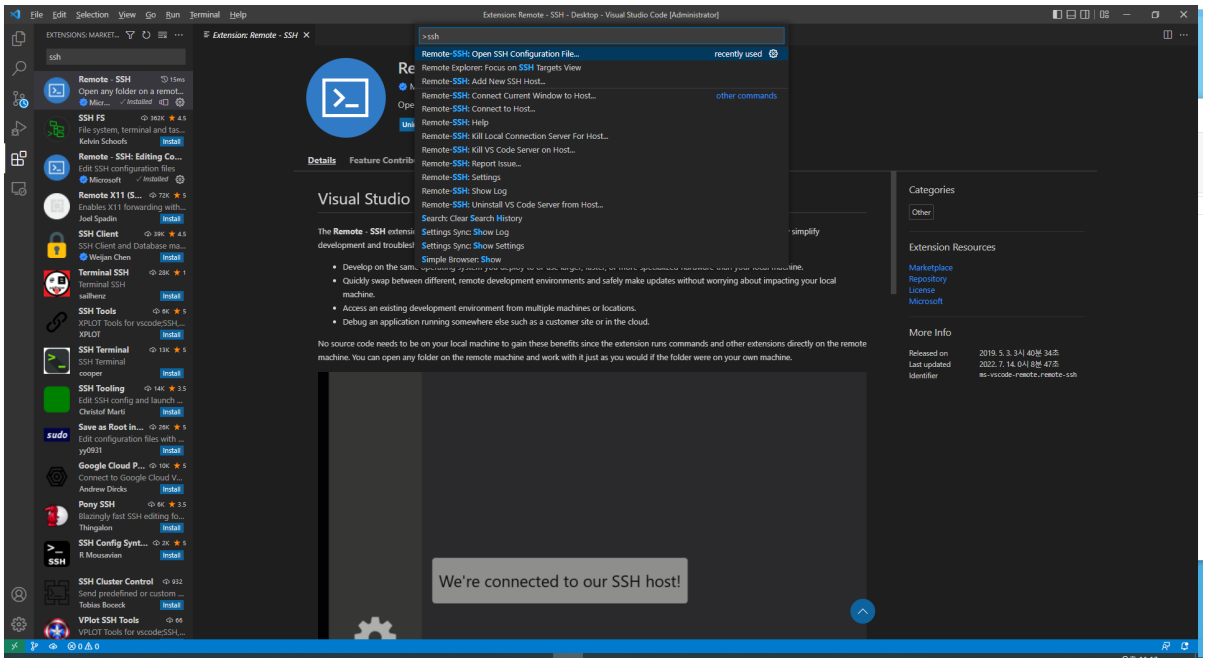


그림 6) ssh 검색

위의 버튼을 눌렀다면 같은 자리에 SSH 구성 파일 리스트가 나열됩니다. 여기서 `C:\Users\<계정명>\.ssh\config`를 선택합니다.

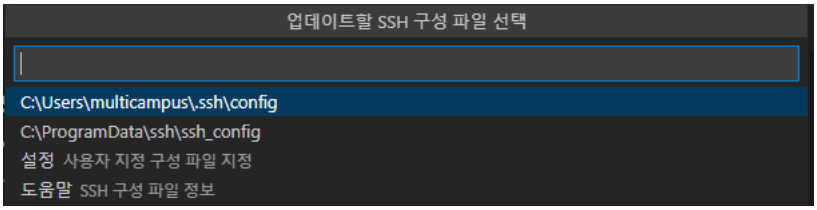


그림 7) SSH 구성 파일 리스트

그러면 SSH 구성 파일이 열립니다.

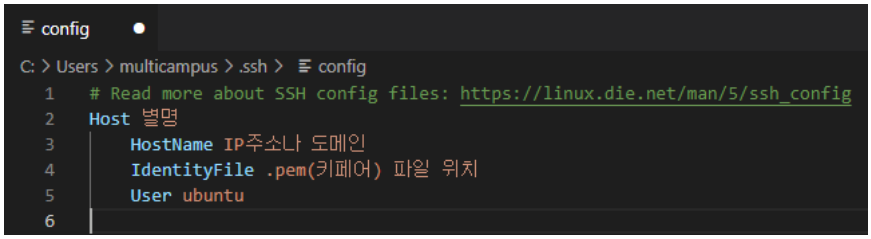


그림 8) SSH 구성 파일

각 요소에 대해 자세히 알아보겠습니다.

- `Host` : `Remote SSH`의 이름을 설정해주면 됩니다. (해당 인스턴스가 무엇인지 알기 쉽게 이름을 정합니다.)
- `HostName` : AWS EC2 인스턴스의 `public IP`나 도메인을 적으면 됩니다.
- `IdentityFile` : 현재 `.pem` 파일이 저장되어있는 위치를 작성하면 됩니다.
- `User` : 계정 이름을 설정한다. 우리는 `ubuntu`를 사용합니다.
- `Port` : 기본값인 22번 포트가 아니라 다른 포트로 ssh 접근을 한다면 입력해줍니다.

3. **SSH 세션 접속**

입력을 다하고 저장한 후 좌측 탭에서 `Remote Explorer` 탭으로 이동합니다.

`SSH TARGET`에 config에서 설정한 Host명으로 아이콘이 하나 생깁니다. Host명 우측의 폴더 아이콘을 클릭합니다.

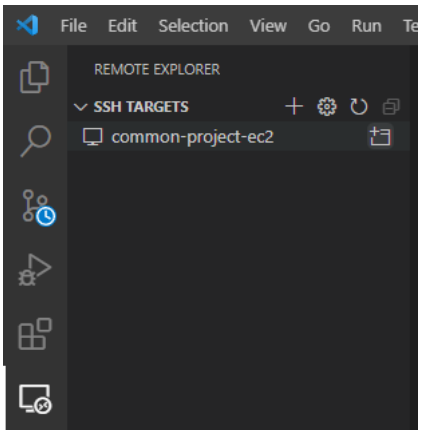


그림 9) Remote Explorer

그럼 새 vscode 창이 뜨면서 Linux, Windows, macOS를 선택하는 창이 나옵니다. 우리는 우분투를 사용하기 때문에 리눅스를 선택합니다.

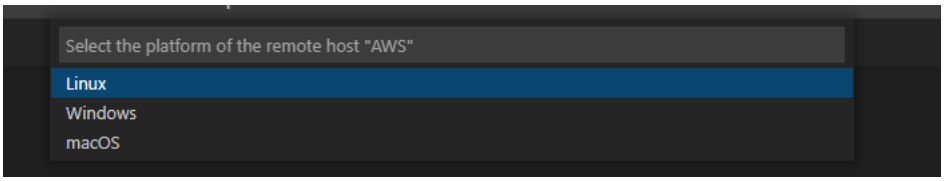


그림 10) AWS 플랫폼 선택창

그러면 SSH를 이용하여 AWS EC2 인스턴스를 vscode에서 편집할 수 있게 됩니다.

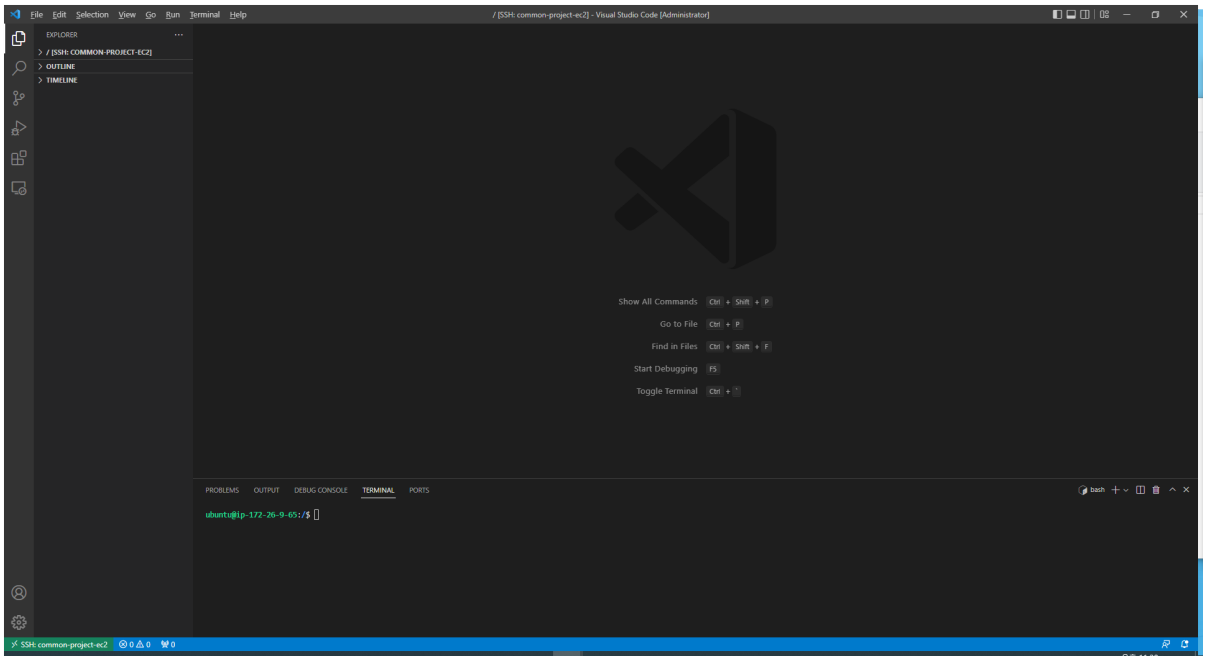


그림 11) Remote SSH에 연결된 모습

4.2. Let’s encrypt 인증서 발급

우리 프로젝트에서는 openvidu를 사용하기 위해 ssl 인증서를 발급받아야 했습니다. 저희는 let’s encrypt 인증서 발급 방식 중 `standalone` 방식을 이용하여 인증서를 발급받았습니다.

이 방식은 80번 포트로 가상 standalone 웹 서버를 띄워 인증서를 발급받는 방식으로 동시에 여러 도메인에 대해 인증서를 발급받을 수 있다는 장점이 있지만 인증서 발급 전에 nginx 서버를 중단해야 한다는 단점이 있습니다.

```
sudo apt update

// letsencrypt 패키지 설치
sudo apt-get install letsencrypt -y

// 실행중인 nginx 종료
service nginx stop

// SSL 인증
certbot certonly --standalone -d uknowme.mooo.com
```

certbot 명령을 실행시켰을 때 만약 인증서가 없다면 인증서를 발급받는 과정을 거칩니다.

```
Saving debug log to /var/log/letsencrypt/letsencrypt.log
Plugins selected: Authenticator standalone, Installer None
Enter email address (used for urgent renewal and security notices) (Enter 'c' to cancel)
:
```

다음은 서비스 약관에 동의하는지 묻습니다. 동의 해줍니다.

```
Please read the Terms of Service at
https://letsencrypt.org/documents/LE-SA-v1.2-November-15-2017.pdf. You must agree in order to register with the ACME server at  htt
ps://acme-v02.api.letsencrypt.org/directory

(A)gree/(C)ancel:
```

다음은 이메일 주소를 공유할 것인지를 묻습니다. 공유한다면 Y, 아니라면 N을 입력하면 됩니다.

```
Would you be willing to share your email address with the Electronic Frontier  Foundation, a founding partner of the Let's Encrypt
 project and the non-profit  organization that develops Certbot? We'd like to send you email about our work  encrypting the web, EF
F news, campaigns, and ways to support digital freedom.

(Y)es/(N)o:
```

인증 완료 후 `certbot certificates` 명령어를 통해 제대로 발급이 되었는지 확인해줍니다.

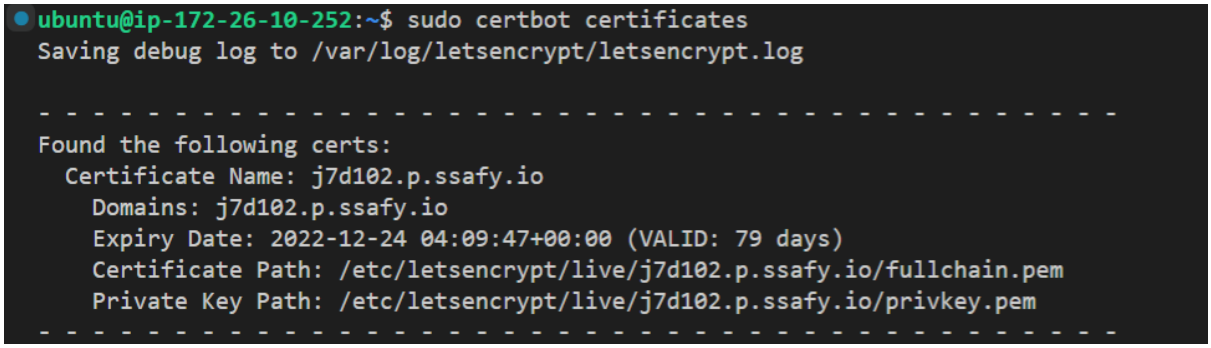


그림 12) letsencrypt 인증서가 발급된 모습

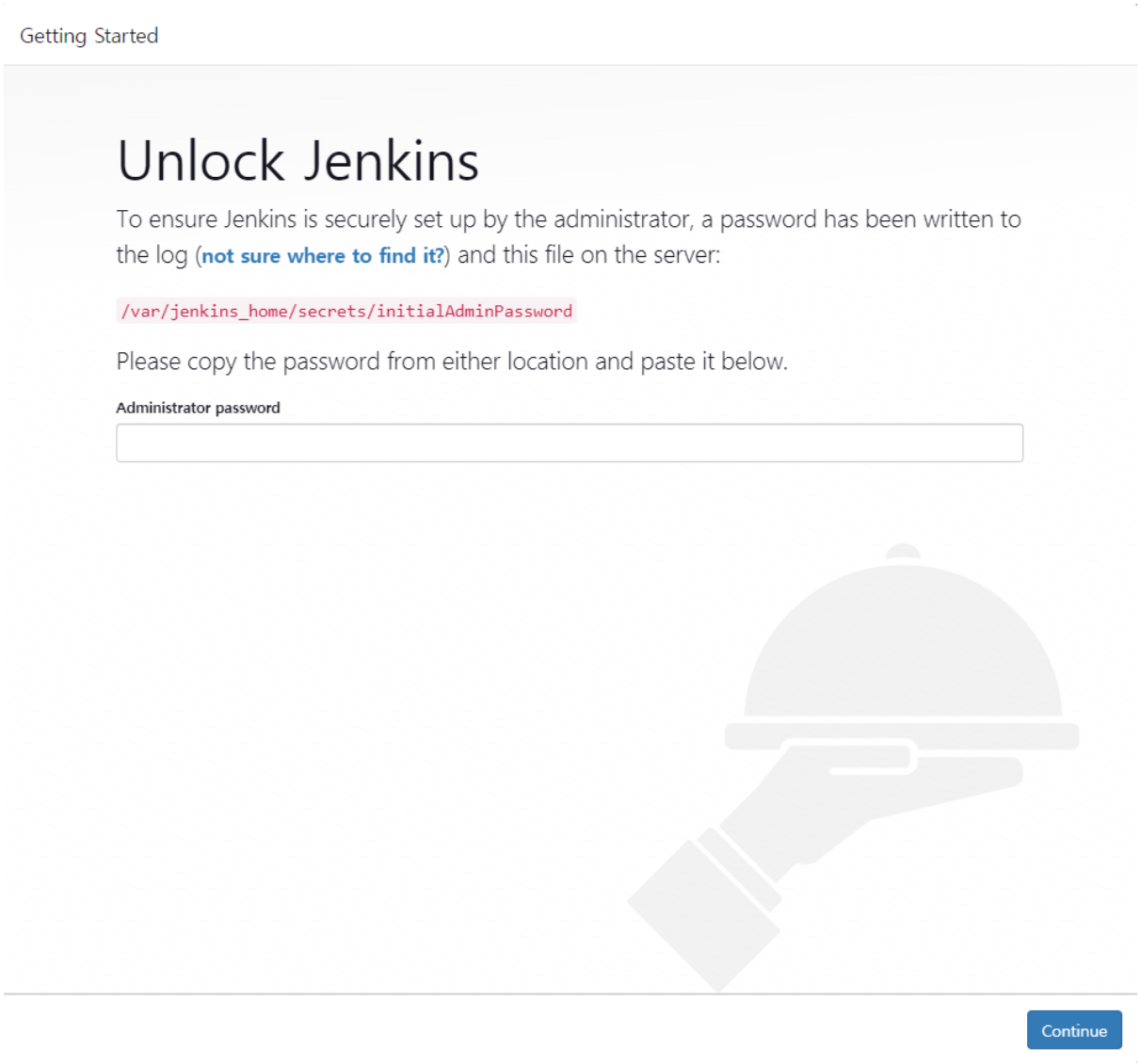
4.3. Jenkins 설정

4.3.1. 컨테이너 실행

우선 Jenkins 컨테이너를 서버에 올려줍니다. 명령어는 다음과 같습니다.

```
docker run --name jenkins-server -itd -p 8088:8080 -v /jenkins:/var/jenkins_home -u root jenkins/jenkins:lts
```

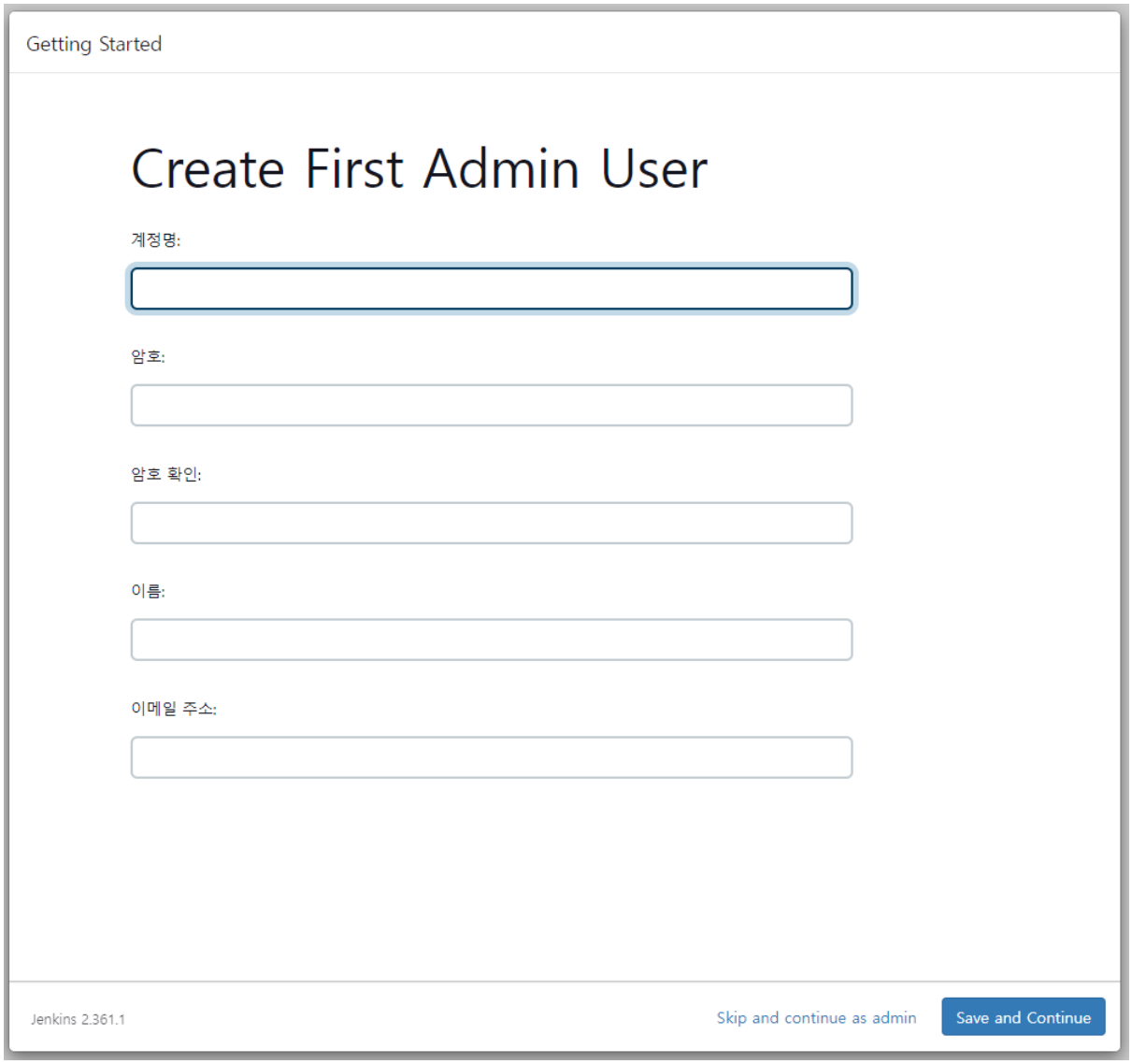
컨테이너를 올린 후 `{hostname}:8088`에 접속한 후 조금 기다리면 패스워드를 입력해달라는 화면이 등장합니다.



이 때 터미널에서 `docker logs jenkins-server`를 입력하면 비밀번호를 확인할 수 있습니다.

```
************************************************************
************************************************************
************************************************************

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

9598d100ec6f40e0a9d6474c1b008aa2

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

************************************************************
************************************************************
************************************************************
```

해당 비밀번호를 입력 후 `Install suggested plugins`를 클릭한 후 플러그인을 다운로드 받아줍니다.



플러그인을 모두 다운로드 받으면, 유저를 등록하는 화면이 등장합니다. 모두 입력해주시고 `Save and Continue` 버튼을 클릭해 넘어갑니다.

그 후 마지막으로 바로 `Save and Finish` 버튼을 누르고 Jenkins 메인 화면으로 넘어갑니다.

4.3.2. 플러그인 설치

왼쪽 탭 `Jenkins 관리`를 클릭 후 `플러그인 관리`로 들어가줍니다. `설치 가능` 탭으로 이동해서 `GitLab`과 `Publish Over SSH`를 체크하고 다운로드 받아줍니다.

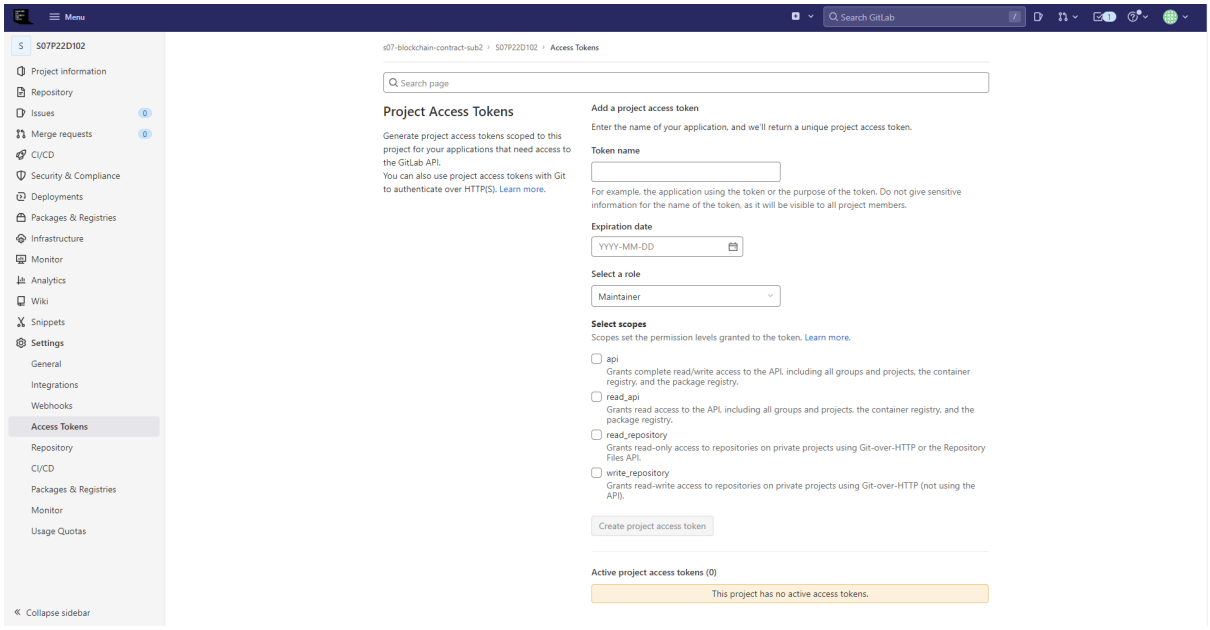
4.3.3. 시스템 설정

- **Gitlab**

왼쪽 탭 `Jenkins 관리`를 클릭 후 `시스템 설정`으로 들어가줍니다.

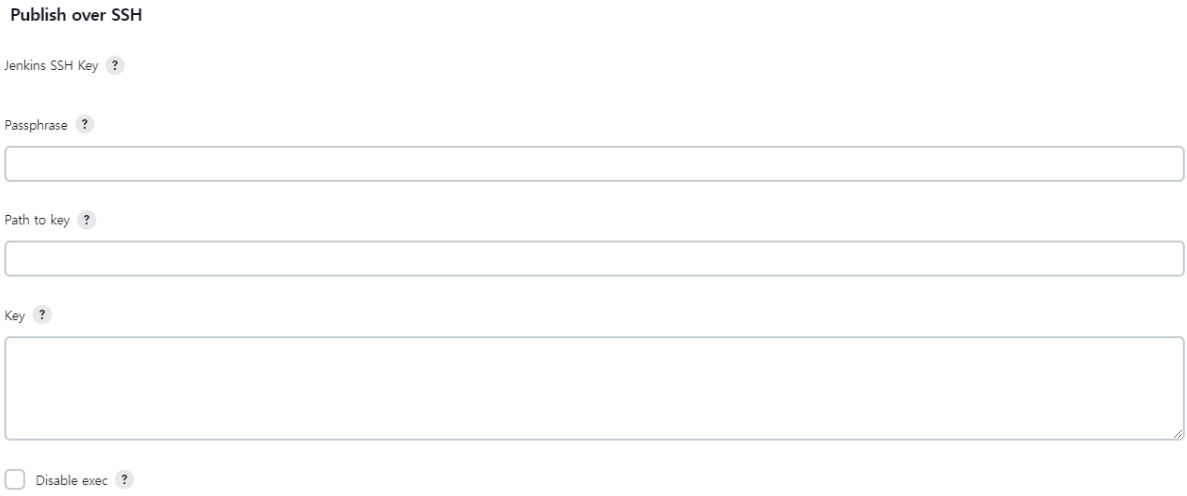
쭉 내려가서 `Gitlab`에서 `Gitlab Connection`을 설정해줍니다. `Connection name`은 원하시는 이름으로 설정하시면 되고 `Gitlab host URL`은 https://lab.ssafy.com, Credential은 Gitlab에서 발급한 Access Token으로 등록해줍니다.

Access Token을 획득하는 방법은 Gitlab Repository 왼쪽 탭에서 `Settings` → `Access Token`에서 발급 받으실 수 있습니다.



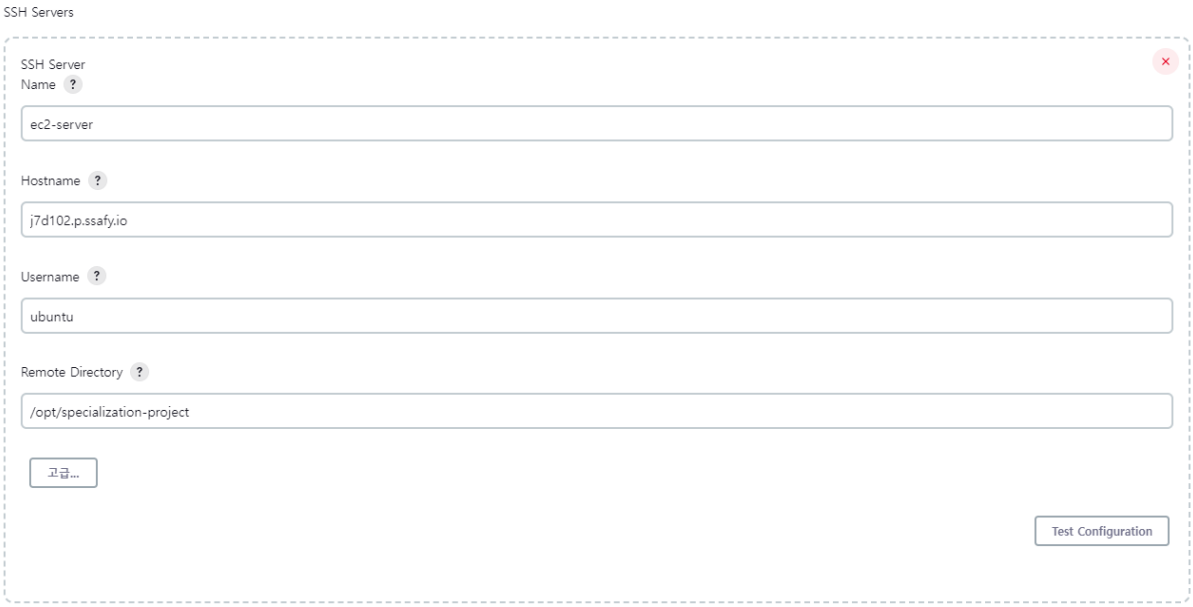
- **Publish over SSH**

Gitlab에서 프로젝트를 가져오고, 빌드한 후 만들어진 자바 압축 파일을 배포하기 위해서는 해당 파일을 도커 컨테이너 내부에서 ec2 서버로 전송해야합니다. 방법은 다음과 같습니다. 우선 `Publish over SSH`으로 이동해줍니다.



`Key`를 입력해줍니다. key는 ssh에 접속할 때 사용하는 키 값을 입력하면 되는데, 저희 같은 경우에는 제공받은 pem 파일의 값을 입력해줍니다.

`SSH Server`에서 추가 버튼을 누릅니다.

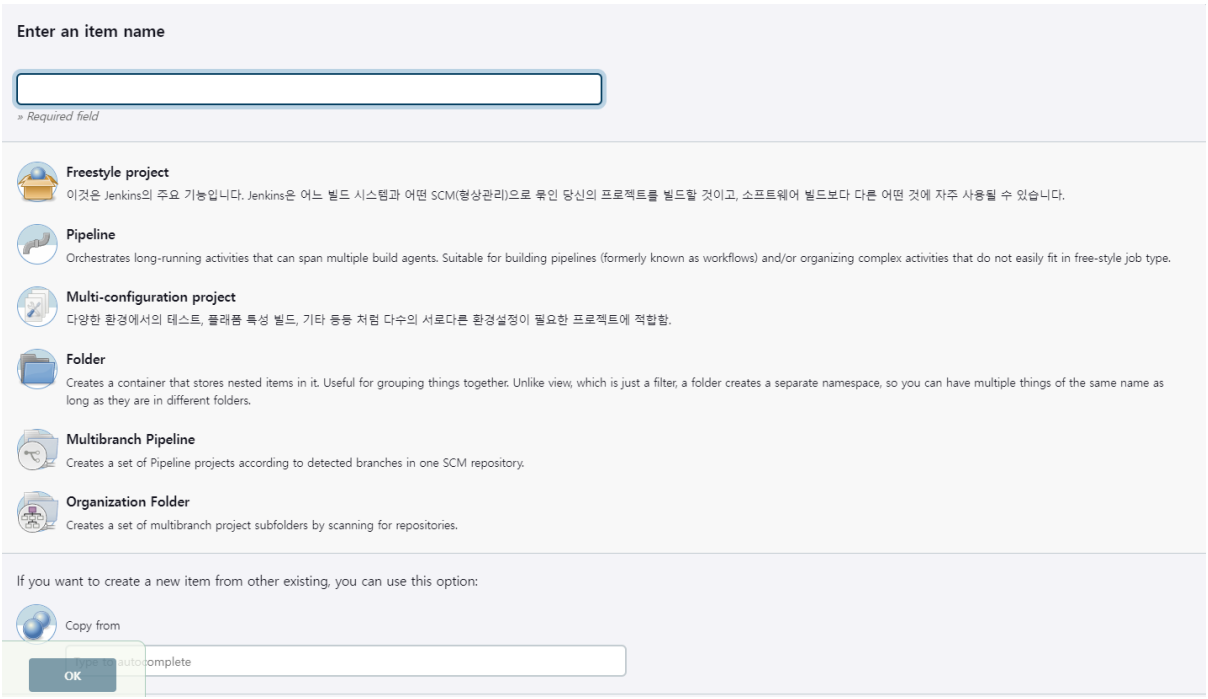


- `Name` : 원하는 값을 입력해주시면 됩니다.
- `Hostname` : ec2 서버의 Public IP 주소나 도메인 네임을 입력해주시면 됩니다.
- `Username` : ec2 서버의 username을 입력해주시면 됩니다. 보통 `ubuntu`로 입력합니다.
- `Remote Directory` : 파일을 전송할 디렉토리 위치를 입력해주시면 됩니다.

4.3.4. Pipeline 생성

- **Item 생성**

Jenkins 메인 화면 왼쪽 탭에서 `새로운 Item`을 클릭합니다.

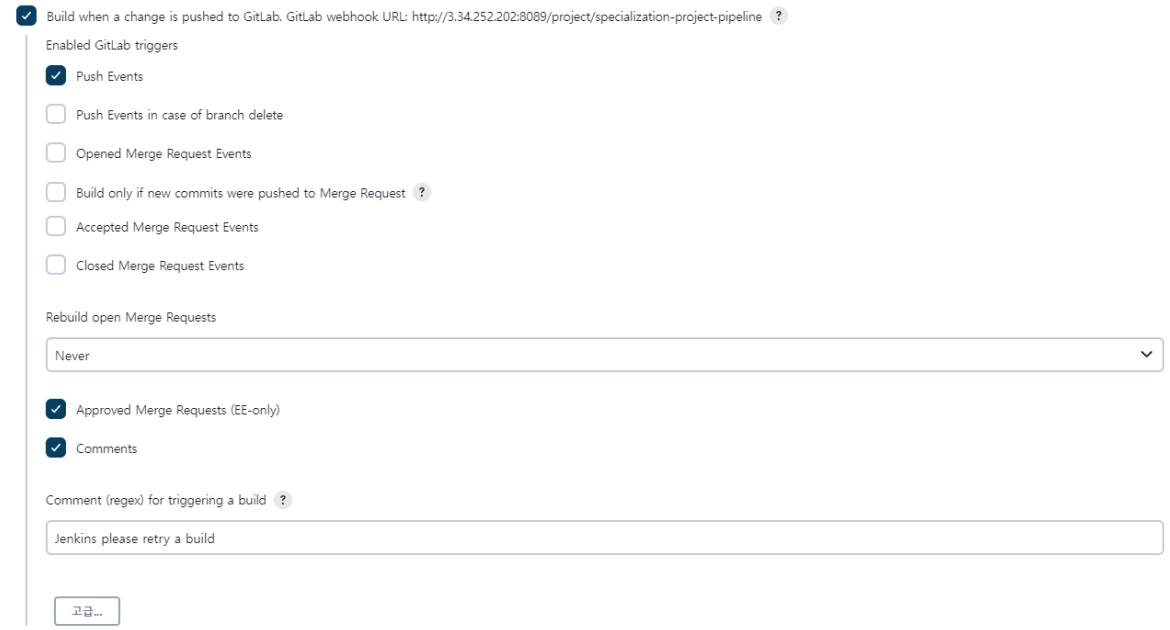


이름을 입력하고 Pipeline을 선택한 후 `OK`를 누릅니다.

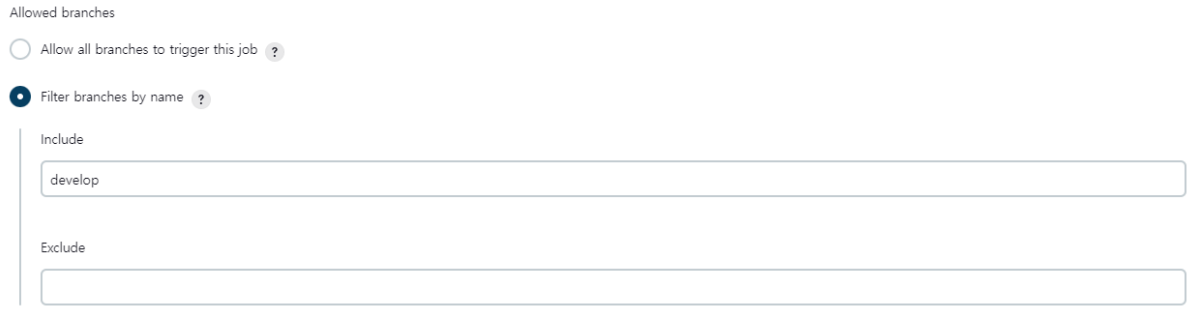
- **GitLab Webhook**

Item이 생성되면 해당 Item을 클릭 후 `구성`으로 들어갑니다.

우선 `Build Triggers`를 설정합니다. `Build when a change is pushed to GitLab. GitLab webhook URL: ~~~`을 체크해줍니다.

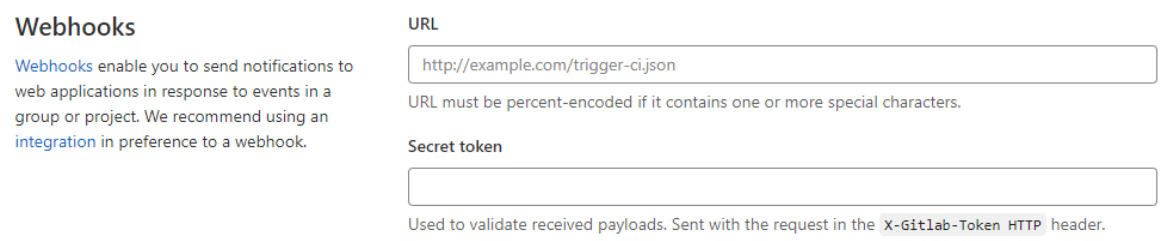


`고급` 버튼을 눌러 `Allowed Branches`에서 `Filter branches by name`을 체크하고 `Include`에서 develop을 입력해줍니다.



그 후 `Secret Token`에서 `Generate` 버튼을 눌러 토큰을 발급받아 줍니다.

그리고 GitLab으로 들어가줍니다. `Settings` → `Webhooks`에서 Push Event가 일어날 경우 웹 훅을 생성해줍니다.



- `URL` : `Build when a change is pushed to GitLab. GitLab webhook URL: ~~~`에서 `~~~` 부분을 입력해줍니다.

- `Secret token` : 방금 발급 받은 Secret Token을 입력해줍니다.

그 후 `Add Webhook`을 클릭하고 등록해줍니다.

- **Pipeline**

파이프라인 스크립트를 입력해줍니다.

```
pipeline {
    agent any

    stages {
        stage('git clone') {
            steps {
                git branch: 'develop', credentialsId: 'gitlab-authentication', url: 'https://lab.ssafy.com/s07-blockchain-contract-
sub2/S07P22D102.git'
            }
        }

        stage('build') {
            steps {
                dir('BE/indive') {
                    sh '''
                        chmod +x gradlew
                        ./gradlew clean build -Pprofile=prod
                    '''
                }
            }
        }

        stage('deploy') {
            steps {
                sshPublisher(publishers: [sshPublisherDesc(configName: 'ec2-server', transfers: [sshTransfer(cleanRemote: false, ex
cludes: '', execCommand: '''docker stop be-server

                docker rm be-server

                cd /opt/specialization-project

                docker build -t mungmnb777/be-server .

                docker run -d -p 8443:443 -p 8080:8080 -v /etc/letsencrypt:/etc/letsencrypt -v /opt/specialization-project/files:/o
pt/specialization-project/files --name be-server mungmnb777/be-server''', execTimeout: 120000, flatten: false, makeEmptyDirs: fals
e, noDefaultExcludes: false, patternSeparator: '[, ]+', remoteDirectory: '', remoteDirectorySDF: false, removePrefix: 'BE/indive/bu
ild/libs', sourceFiles: 'BE/indive/build/libs/indive-0.0.1-SNAPSHOT.jar')], usePromotionTimestamp: false, useWorkspaceInPromotion:
 false, verbose: false)])
            }
        }
    }
}
```

4.4. MySQL 컨테이너 생성

다음 명령어로 MySQL 컨테이너를 생성합니다.

```
docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=<password> -d -p 3306:3306 mysql:latest
```

workbench를 이용해 `{hostname}:3306`으로 해당 DB 서버에 접속한 후 `create database indive`를 통해서 데이터베이스를 생성해줍니다.

추가로 저희는 테스트용 MySQL 서버를 만들어두었습니다.

```
docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=<password> -d -p 33066:3306 mysql:latest
```

4.5. Redis 컨테이너 생성

다음 명령어로 Redis 컨테이너를 생성합니다.

```
docker run --name redis-server -d -p 6379:6379 redis
```

4.6. IPFS 컨테이너 생성

1. Golang 설치

```
sudo apt install golang -y
```

2. IPFS 도커 이미지 가져오기

```
docker pull ipfs/go-ipfs
```

3. 개인 체인 키 생성

```
go get -u github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen

cd ~/go/src/github.com/Kubuxu/go-ipfs-swarm-key-gen/ipfs-swarm-key-gen

go build

./ipfs-swarm-key-gen > ~/.ipfs/swarm.key
```

4. 디렉토리 생성

```
mkdir -p ~/ipfs/ipfs1/data ~/ipfs/ipfs1/export ~/ipfs/ipfs2/data ~/ipfs/ipfs2/export
```

5. 체인 키 복사

```
cp ~/.ipfs/swarm.key ~/ipfs/ipfs1/data

cp ~/.ipfs/swarm.key ~/ipfs/ipfs2/data
```

6. 컨테이너 실행

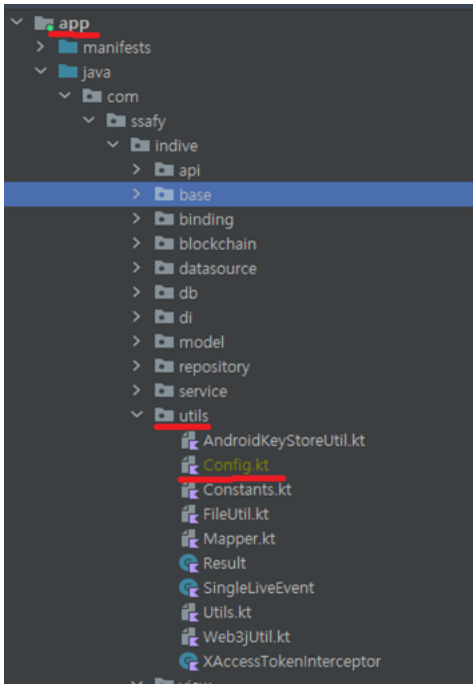
```
docker run -d --name ipfs_host1 --privileged=true --entrypoint="sh" -v ~/ipfs/ipfs1/export:/export -v ~/ipfs/ipfs1/data:/data/ipfs
 -p 4001:4001 -p 8085:8080 -p 5001:5001 ipfs/go-ipfs:latest -c "while true; do sleep 1;done"

docker run -d --name ipfs_host2 --privileged=true --entrypoint="sh" -v ~/ipfs/ipfs2/export:/export -v ~/ipfs/ipfs2/data:/data/ipfs
 -p 4011:5001 -p 9085:8080 -p 5011:5001 ipfs/go-ipfs:latest -c "while true; do sleep 1;done"
```

안드로이드

https://drive.google.com/file/d/134IjReijJCmbJ-ZV9Y_XeP6P6QGQ5s3t/view?usp=sharing

<! — 블록체인 네트워크 서버가 불안전해 컨트랙트 주소가 바뀔 수 있음 —>

1. Config.kt



App -> java -> com.ssafy.indive -> utils 에 Config.kt 추가

블록체인 네트워크

1. 프라이빗 블록체인 네트워크 구성

1.1 패키지 설치

리눅스 환경에서 이더리움 네트워크를 만들기 위해 필요한 패키지를 설치해줍니다.

```
$ sudo apt-get install software-properties-common
$ sudo add-apt-repository -y ppa:ethereum/ethereum
$ sudo apt-get update
$ sudo apt-get install ethereum
```

2022-09-14 기준 1.10.23-stable 버전을 사용했습니다.

```
$ geth version

Geth
Version: 1.10.23-stable
Git Commit: d901d85377c2c2f05f09f423c7d739c0feecd90a
Architecture: amd64
Go Version: go1.18.5
Operating System: linux
GOPATH=
GOROOT=go
```

1.2 첫 번째 블록 생성

연결된 블록을 형성하기 위해 첫 번째 블록이 필요합니다. Genesis Block 이라고 하며 블록체인 네트워크에 대한 설정을 해줄 수 있습니다.

제네시스 블록은 아래와 같은 세팅을 사용했습니다. 괄호 안의 값은 일단 바꾸지 않고 다음으로 진행합니다.

- Genesis.json

```
{
  "config": {
    "chainId": 102,
    "homesteadBlock": 0,
    "eip150Block": 0,
    "eip155Block": 0,
    "eip158Block": 0,
    "byzantiumBlock": 0,
    "constantinopleBlock": 0,
    "petersburgBlock": 0,
    "istanbulBlock": 0,
    "berlinBlock": 0,
    "contractSizeLimit": 2147483647,
    "clique": {
      "period": 1,
      "epoch": 30000
    }
  },
  "difficulty": "0x400",
  "nonce" : "0x0000000000000042",
  "gasLimit": "0x1fffffffffffff",
  "extradata": "0x0000000000000000000000000000000000000000000000000000000000000000{0x를 제외한 주소}00000000000000000000000000000000000000000000000000000000000000000",
  "coinbase" : "{주소}",
  "mixhash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "parentHash" : "0x0000000000000000000000000000000000000000000000000000000000000000",
  "timestamp" : "0x00",
  "alloc": {
    "{주소}": { "balance": "1000000000000000000000000" }
  }
}
```

- config : geth 1.6 이전 버전 접속 시 발생할 수 있는 문제점을 해결하는 옵션. 사설 네트워크 일 결우 위 값으로 설정
  - chainId : 현재 chain을 식별하는 값. replay attack을 막기 위해 사용
  - homesteadBlock : Homestead는 두 번째 Ethereum release 버전 0으로 set하면 Homestead 버전을 사용하는 것을 의미
  - eip150Block : Ethereum Improvement Proposal; 이더리움 플랫폼 표준 문서
- mixhash : nonce와 함께 블록의 작업증명을 위한 옵션
- parentHash : nonce와 mixhash를 포함한 부모 블록의 헤더에 대한 해시 값을 갖는 옵션. Genesis 블록은 최초의 블록이기 때문에 값은 0입니다.
- difficulty : 블록 생성을 위한 계산 난이도 설정. 값에 따라 채굴 속도가 달라짐
- gasLimit : 하나의 블록이 담을 수 있는 gas의 임계치. 값을 높일 수록 block 하나에 포함할 수 있는 gasLimit이 늘어남
- coinbase : 해당 블록에 대해 채굴에 성공하면 얻게되는 총 보상금을 160비트 주소값으로 표현. 채굴 보상금과 스마트 컨트랙트 실행의 환불 값의 합을 나타내며 benefitciary 또는 etherbase라고도 함.
- extraData : PoA validators/sealers 가 기록되는 영역.
  - 32byte 공간에 채워진 0
  - 0x 접두사를 제외한 validators/sealers 들의 연결된 주소
  - 65byte 공간에 채워진 0
- coinbase : 블록 채굴시 보상이 전송되는 160bit 주소. 제네시스 블록은 어떤 값이든 될 수 있음
  - 채굴 보상 + 계약 거래 환불 합
- timestamp : 해당 블록이 취득된 시점을 나타내는 옵션. Genesis 블록은 0
  - 용도 - (1)작업증명의 난이도 조절 (2) 블록 간 순서 확인
- alloc : genesis 블록 생성과 함께 이더리움 계좌에 원하는 액수의 이더를 미리 송금 가능. 계좌의 주소와 할당량을 지정 가능하며 wei 단위

1.3 계정 생성

genesis 블록 초기화 전 계정을 생성합니다. 제네시스 블록의 alloc 에서 이더를 미리 넣어 놓을 주소입니다. 여기서 생성되는 계정의 주소를 genesis 블록의 alloc, extraData, coinbase 에 모두 작성합니다.

비밀번호를 입력해야하는데, 계정 잠금 및 비밀키 복호화를 위해 필요한 번호이니, 잃어버리지 않도록 합니다.

```
geth --datadir ./data account new
```

1.4 네트워크 초기화

제네시스 블록으로 네트워크를 초기화 한다. keystore 등 데이터는 현재경로/data 에 저장됩니다.

```
geth --datadir ./data init genesis.json
```

1.5 서버 실행

서버 배포를 시작합니다. 설정 값은 아래에서 확인할 수 있습니다.

“nohup {아래 명령어} & “ 를 사용하면 백그라운드에서 동작하며, 터미널을 닫아도 서버는 계속 실행됩니다. 로그 확인은 1.5.1 번을 확인하면 됩니다.

```
geth --datadir ./data --networkid 102 --nodiscover --http --http.api "db,personal,eth,net,web3,debug,miner" --http.corsdomain="*" --ht
```

- nohup … &: 백그라운드 실행
  - console 명령어 제거
  - 종료 : kill -15 PROCCESS ID
  - 상태 확인 : ps -ef | grep geth
- —datadir ./data : geth 데이터가 저장될 경로
- networkid : 네트워크 식별자
- nodiscover : 같은 제네시스 블록과 네트워크 ID에 있는 블록들이 연결되는 것을 방지
- http
  - api : http 로 접근 가능한 api 명령어
  - corsdomain : 접속가능한 http 도메인 지정.
  - addr : 현재 사용중인 IP
  - port : 서버 포트. 기본값 8545
  - vhost : 허용할 가상 호스트 도메인. 기본값 localhost
- console : 명령어를 통해 실행
- mine : 채굴
- `--miner.gasprice 0 --miner.gaslimit 0`

1.5.1 백그라운드 서버 확인

- 구동 상태 확인

geth 가 현재 어떤 포트에서, 그리고 어떤 명령어로 실행 됐는지 확인할 수 있습니다.

```
ps -ef | grep geth
```

- 서버 로그 출력

백그라운드 프로세스로 동작중인 geth의 로그는 nohup.out 파일에 저장됩니다. 아래 명령어를 실행하면, 마지막 10라인을 실시간으로 출력합니다.

```
tail -f nohup.out
```

1.6 채굴 시작

채굴을 시작하기 위해 계정 잠금을 해제 해줍니다.

```
> personal.unlockAccount(eth.accounts[0])
> miner.start()
```

1.6.1 채굴 관련 함수

```
> miner.start() // 채굴 시작
> miner.stop() // 채굴 종료
> eth.mining // 채굴 상태 확인 (T/F)
```

2. 컨트랙트 배포

2.1 컨트랙트 코드

- InDive.sol

```
// SPDX-License-Identifier: UNLICENSED
pragma solidity ^0.8.4;

import "./token/ERC20/InDiveToken.sol";
import "./token/ERC20/IERC20.sol";
import "./token/ERC721/InDiveNFT.sol";
import "./token/ERC721/IERC721.sol";
import "./utils/Strings.sol";

contract Indive {

    using Strings for *;

    // 토큰 컨트랙트 주소
    address _InDiveTokenAddress;
    // 토큰 컨트랙트
    IERC20 _InDiveTokenContract;

    uint donationSeq;

    event DonationEvent(address indexed artist, address indexed donator, uint256 value, string message, uint256 time);

    // donationHistory[주소] = 후원 및 후원 받은 기록 리스트
    mapping(address => DonationHistory[]) public donationHistory;

    // donatorList[아티스트 주소] = 후원자 리스트
    mapping(address => DonationInfo[]) public donatorList;

    // ranking[아티스트 주소][사용자 주소] = 후원
    mapping(address => mapping(address => bool)) public isDonated;

    struct DonationInfo{
        address addr;
        uint256 totalValue;
    }

    struct DonationHistory{
        uint seq;
        address addr;
        string state;
        uint256 value;
        string message;
        uint256 time;
    }

    constructor () {
        donationSeq = 0;
    }
```

```
    // IVE 토큰 컨트랙트 연결
    function setTokenContract(address contractAddress) public {
        _InDiveTokenAddress = contractAddress;
        _InDiveTokenContract = IERC20(_InDiveTokenAddress);
    }

    // donate : 컨트랙트 호출자가 to 에게 토큰을 송금한다.
    // 사용자는 InDiveToken 컨트랙트에서 approve를 사용하여 InDive 컨트랙트에 출금할 권한을 준다.
    // InDive 컨트랙트는 토큰을 출금 후 to 에게 송금한다.
    function donate(address _to, uint256 _value, string memory _message, uint256 _time) public returns (bool) {
        // require(_value >= 2, "donate value error!");

        // _InDiveTokenContract.transferFrom(msg.sender, address(this), _value);
        // _InDiveTokenContract.approve(address(this), _value - 1);
        // // 컨트랙트에 저장된 토큰을 to 에게 전송한다.
        // _InDiveTokenContract.transferFrom(address(this), _to, _value - 1);

        _InDiveTokenContract.transferFrom(msg.sender, address(this), _value);
        _InDiveTokenContract.approve(address(this), _value);
        // 컨트랙트에 저장된 토큰을 to 에게 전송한다.
        _InDiveTokenContract.transferFrom(address(this), _to, _value);

        // 배열에 후원 값을 저장한다.
        // 후원한 적이 있다면 후원 내용을 갱신한다.
        if(isDonated[_to][msg.sender]){
            for(uint i ; i < donatorList[_to].length ; i++){
                if(donatorList[_to][i].addr == msg.sender){
                    donatorList[_to][i].totalValue += _value;
                    break;
                }
            }
        } else {
            donatorList[_to].push(DonationInfo(msg.sender, _value));
            isDonated[_to][msg.sender] = true;
        }

        // _to 배열에 송금 받은 내용을 기록한다.
        donationHistory[_to].push(DonationHistory(donationSeq, msg.sender, "Get", _value, _message, _time));
        // 사용자 배열에 송금 내용을 기록한다.
        donationHistory[msg.sender].push(DonationHistory(donationSeq, _to, "Send", _value, _message, _time));

        donationSeq += 1;

        // 후원 내용을 블록에 기록한다.
        emit DonationEvent(_to, msg.sender, _value, _message, _time);

        return true;
    }

    function getDonatorList(address artist) public view returns (string memory){
        string memory result = "[";

        for(uint i = 0 ; i < donatorList[artist].length ; i++){
            address addr = donatorList[artist][i].addr;
            uint256 totalValue = donatorList[artist][i].totalValue;

            string memory strAddr = Strings.toHexString(addr);
            string memory strValue = Strings.toString(totalValue);

            result = string(abi.encodePacked(result, "{", "\"address\":\"", strAddr, "\",\"totalValue\":", strValue,"}"));

            if(i < donatorList[artist].length - 1){
                result = string(abi.encodePacked(result, ","));
            }
        }

        result = string(abi.encodePacked(result, "]"));

        return result;
    }

    function getDonationHistoryList(address artist) public view returns (string memory){
        string memory result = "[";

        for(uint i = 0 ; i < donationHistory[artist].length ; i++){
            uint256 seq = donationHistory[artist][i].seq;
            address addr = donationHistory[artist][i].addr;
            string memory state = donationHistory[artist][i].state;
            uint256 value = donationHistory[artist][i].value;
            string memory message = donationHistory[artist][i].message;
            uint256 time = donationHistory[artist][i].time;

            result = string(abi.encodePacked(result, "{\"seq\":", Strings.toString(seq), ",\"address\":\"", Strings.toHexString(addr),

            if(i < donationHistory[artist].length - 1){
                result = string(abi.encodePacked(result, ","));
            }
```

```
        }

        result = string(abi.encodePacked(result, "]"));

        return result;
    }
}
```

- InDiveToken.sol

```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./ERC20.sol";

contract InDiveToken is ERC20 {
    uint public INITIAL_SUPPLY = 1000000;

    constructor() ERC20("InDive Token", "IVE"){
        _mint(msg.sender, INITIAL_SUPPLY);
    }

    function addressOfContract() public view returns (address) {
        return address(this);
    }
}
```

- InDiveNFT.sol

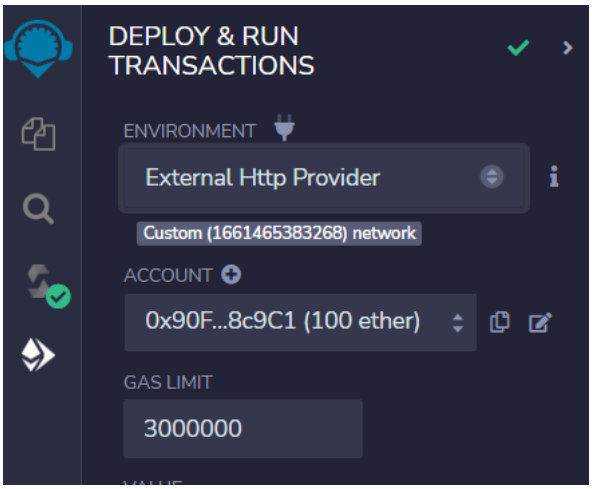
```
// SPDX-License-Identifier: MIT
pragma solidity ^0.8.0;

import "./ERC721.sol";
import "./extensions/ERC721URIStorage.sol";
import "./extensions/ERC721Enumerable.sol";
import "../../utils/Counters.sol";

contract InDiveNFT is ERC721, ERC721URIStorage, ERC721Enumerable {
    // 토큰 Id를 관리할 객체
    using Counters for Counters.Counter;
    Counters.Counter private _tokenIds;

    address public owner;

    // NFT를 발급하기 위한 최소 후원 값
    mapping(address => uint256) minNFTValue;

    constructor () ERC721("InDiveNFT", "IVEN") {
        owner = msg.sender;
    }

    // NFT 제작
    function safeMint(address _to, string memory _tokenURI) public returns (uint256) {

        // 새로운 토큰 Id 값 증가
        _tokenIds.increment();
        uint256 newItemId = _tokenIds.current();

        // nft 제작
        _mint(_to, newItemId);
        // URI 저장
        _setTokenURI(newItemId, _tokenURI);

        return newItemId;
    }

    // 해당 주소가 소유한 NFT들을 조회한다.
    function getNFTTokens(address _nftTokenOwner) view public returns (string[] memory) {

        uint256 balanceLength = balanceOf(_nftTokenOwner);
        string[] memory nftURIList = new string[](balanceLength);

        for(uint256 i = 0; i < balanceLength; i++){
          uint256 nftTokenId = tokenOfOwnerByIndex(_nftTokenOwner, i);
          string memory nftTokenURI = tokenURI(nftTokenId);
            nftURIList[i] = nftTokenURI;
        }
        return nftURIList;
    }

    // tokenId 로 tokenURI 반환
```

```
    function tokenURI(uint256 tokenId) public view override(ERC721, ERC721URIStorage) returns (string memory) {
        return super.tokenURI(tokenId);
    }

    // tokenId 에 해당하는 NFT 삭제
    function _burn(uint256 tokenId) internal override(ERC721, ERC721URIStorage) {
        super._burn(tokenId);
    }

    function _beforeConsecutiveTokenTransfer(
        address,
        address,
        uint256,
        uint96 size
    ) internal virtual override(ERC721, ERC721Enumerable) {
        // We revert because enumerability is not supported with consecutive batch minting.
        // This conditional is only needed to silence spurious warnings about unreachable code.
        if (size > 0) {
            revert("ERC721Enumerable: consecutive transfers not supported");
        }
    }

  function _beforeTokenTransfer(address from, address to, uint256 tokenId) internal override(ERC721, ERC721Enumerable) {
    super._beforeTokenTransfer(from, to, tokenId);
  }

  function supportsInterface(bytes4 interfaceId) public view override(ERC721, ERC721Enumerable) returns (bool) {
    return super.supportsInterface(interfaceId);
  }

  function curruntTokenId() public view returns (uint256){
      return _tokenIds.current();
  }
}
```

그 외 파일도 함께 Remix에 넣고 배포는 위 세 개의 파일만 진행하면 됩니다.

2.2 컨트랙트 배포

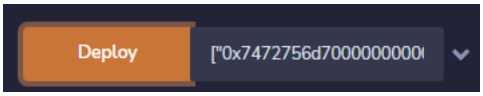
RemixIDE 환경에서 배포합니다. ENVIRONMENT를 External Http Provider로 설정 후 포트를 포함한 서버 주소를 입력하여 연결을 확인합니다.



서버에서 계정 언락을 진행합니다.

```
personal.unlockAccount(eth.accounts[0])
```

컴파일된 솔리디티 파일을 선택 후 Deploy를 눌러 배포합니다.

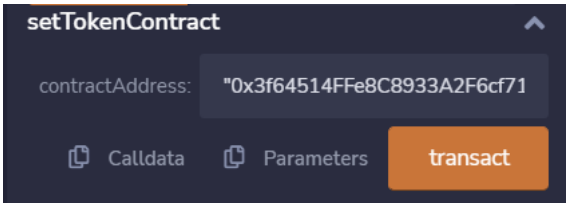


트랜잭션 결과

- Remix IDE

2.3 컨트랙트 연결

InDive 컨트랙트의 setTokenContract에 배포된 InDiveToken 컨트랙트의 주소를 입력 후 실행합니다.



3. Web3j Wrapper Class 생성

3.1 개요

Solidity 함수를 Java 코드에서 실행하기 위해 Wrapper Class가 필요합니다. 반드시 필요한 것은 아니지만, 함수를 간편하게 실행할 수 있습니다.

윈도우 10 환경을 기준으로

3.2 Web3j 설치

- https://github.com/web3j/web3j

관리자 권한으로 Powershell 실행 후 입력합니다.

```
Set-ExecutionPolicy Bypass -Scope Process -Force; iex ((New-Object System.Net.WebClient).DownloadString('https://raw.githubusercontent
```

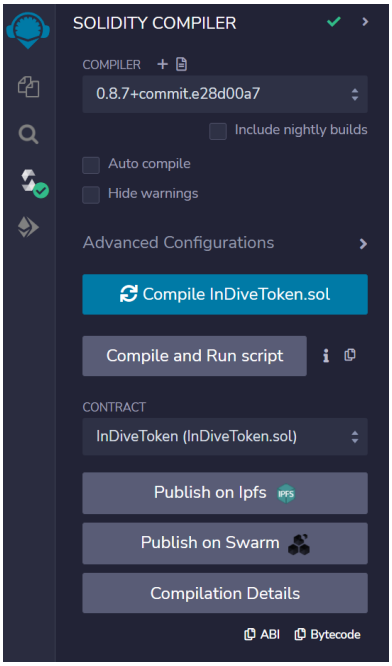
3.3 버전 확인

CMD 에서 확인 합니다.

```
web3j -v
```

3.3 ABI 파일 다운로드

Remix에서 Solidity 파일 컴파일 후 ABI 를 복사합니다. 복사한 ABI 는 C:\web3j\abi 폴더에 텍스트 파일로 저장합니다. 다른 경로에 저장하여도 무방합니다.



3.4 Wrapper Class 생성

- 필요한 옵션
  - -a : abi 파일이 저장된 경로. 파일명까지 지정해야함
  - -o : 래퍼 클래스 파일이 저장될 경로
  - -p : 사용할 프로젝트의 패키지명.
- -b 는 변환할 컴파일러 경로를 지정하는데, 윈도우에 web3j를 설치해줬기 때문에 지정할 필요가 없다.

```
$ web3j generate solidity [-hV] [-jt] [-st] -a=<abiFile> [-b=<binFile>] -o=<destinationFileDir> -p=<packageName>

   -h, --help                        Show this help message and exit.
   -V, --version                     Print version information and exit.
   -jt, --javaTypes                  use native java types. Default: true
   -st, --solidityTypes              use solidity types.
   -a, --abiFile=<abiFile>           abi file with contract definition.
   -b, --binFile=<binFile>           optional bin file with contract compiled code in order to generate deploy methods.
   -o, --outputDir=<destinationFileDir> destination base directory.
   -p, --package=<packageName>       base package name.
```

- 예시 코드

```
> web3j generate solidity -a C:\web3j\abi\InDive.txt -o C:\web3j\result -p com.ssafy.indive
             _      _____ _
            | |    |____ (_)
__      _____| |__      / /_
\ \ /\ / / _ \ '_ \     \ \ |
 \ V  V /  __/ |_) |.___/ / |
  \_/\_/ \___|_.__/ \____/| |
                        _/ |
                       |__/
by Web3Labs
Generating com.ssafy.indive.InDive ... File written to C:\web3j\result
```

4. Keystore → Private Key

- geth 로 구축된 블록체인 네트워크의 keystore 파일에서 Private Key로 변환합니다.
- Ubuntu 환경에서 진행합니다.

4.1 APT 패키지 매니저로 모듈 설치

- curl 설치
  - 서버와 통신할 수 있는 커맨드 명령어 툴

```
$ sudo apt-get install -y curl
```

- nodejs 설치 및 버전 확인

```
$ sudo apt update
$ sudo apt install nodejs
$ nodejs -v
```

- npm 설치

```
$ sudo apt install npm
```

4.2 Ethereumjs-Wallet 모듈 설치

- keystore 파일을 private key로 변환해줄 모듈
- 0.6.5를 설치해야합니다.

```
$ npm install ethereumjs-wallet@0.6.5
```

4.3 Javascript 코드 작성

- 블록체인 서버에서 빈 파일 생성 후 아래 코드를 작성한 후 저장합니다.
- getPrivateKey.js

```
const Wallet = require('ethereumjs-wallet'),
      fs = require('fs');


const utcFile = "[KeyStore 파일 경로]"
const password = "[해당 계정 생성할 때 작성한 비밀번호]"


const myWallet = Wallet.fromV3(fs.readFileSync(utcFile).toString(), password, true);

console.log("Private Key: " + myWallet.getPrivateKey().toString('hex'))
console.log("Address: " + myWallet.getAddress().toString('hex'))
```

4.4 실행

- 작성한 코드를 node로 실행합니다.
- 경로가 정확하게 입력되어야 합니다. 루트 폴더에 있는 경우 경로를 입력하지 않아도 됩니다.

```
$ node getPrivateKey.js
```

4.5 결과

- 어느정도 시간이 지난 후 복호화된 Private Key와 Address를 출력합니다.

```
Private Key: {비밀키}
Address: {주소}
```