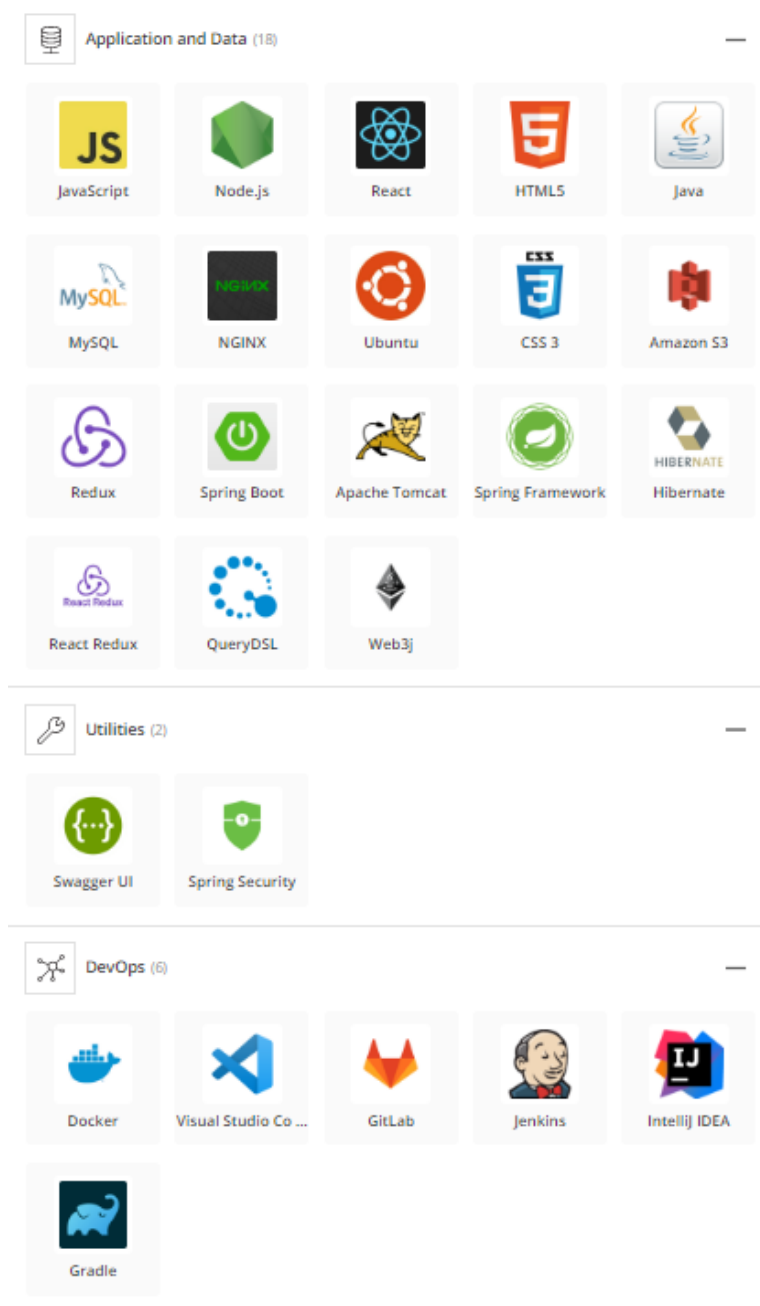




# DON-JO 포팅 메뉴얼

## 1. 프로젝트 구성도

### 기술 스택(stackshare)



## 시스템 아키텍처

## 2. Spring 서버 ec2 세팅

Server spec : t2.micro

os : Amazon Linux 2

### docker 설치

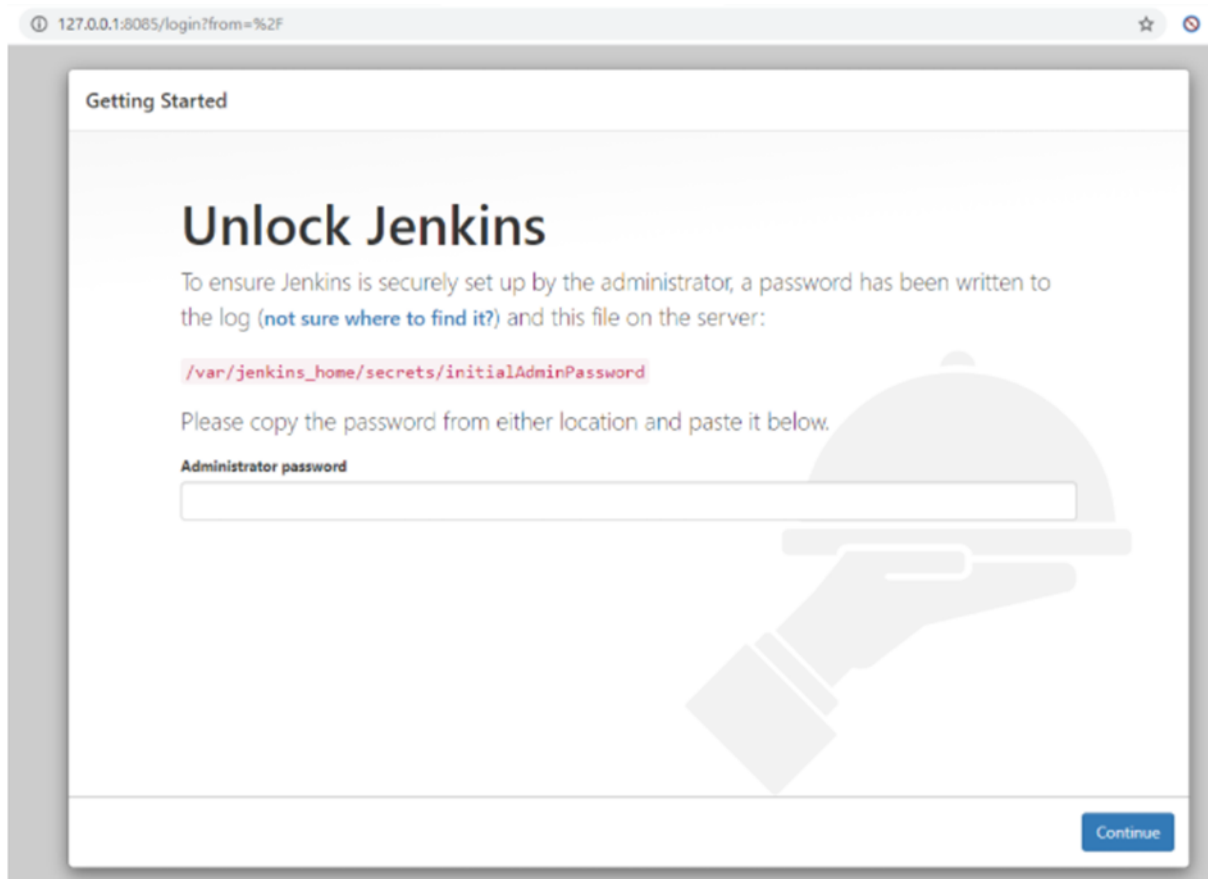
```
// docker 설치

// 패키지 업데이트
sudo apt update
// https관련 패키지 설치
sudo apt install apt-transport-https ca-certificates curl software-properties-common
// docker repository 접근을 위한 gpg 키 설정
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo apt-key add -
// docker repository 등록
sudo add-apt-repository "deb [arch=amd64] https://download.docker.com/linux/ubuntu focal stable"
// 다시 업데이트
sudo apt update
// 도커 설치
sudo apt install docker-ce
// 설치 확인
docker --version
```

## 3. jenkins 서버 ec2 세팅

```
// Jenkins 설치

// 젠킨스 이미지 다운로드
docker pull jenkins/jenkins:lts
// 젠킨스 컨테이너 설치 및 실행
docker run -itd -p 8080:8080 -v /jenkins:/var/jenkins_home -name jenkins -u root jenkins/jenkins:lts
```



```
// 젠킨스 컨테이너에 접근해서 어드민 패스워드 찾기  
  
docker exec -it jenkins /bin/bash  
  
cat /var/jenkins_home/secrets/initialAdminPassword  
  
// 혹은 jenkins 로그를 출력해서 초기 비번 확인  
docker logs jenkins -f
```

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

Install plugins the Jenkins community finds most useful.

Select and install plugins most suitable for your needs.

## Getting Started

© 2015 Pearson Education, Inc. or its affiliate(s). All rights reserved. Pearson Education, Inc., publishing as Pearson Benjamin Cummings, 101 Philip Drive, Assinippi Park, New York, NY 10984-2135

|                  |                                 |                                     |                        |   |
|------------------|---------------------------------|-------------------------------------|------------------------|---|
| ✓ Folders Plugin | ✓ OWASP Markup Formatter Plugin | ✓ Build Timeout                     | ⌚ Credentials Binding  | ** SSH server<br><b>Folders</b><br><b>OWASP Markup Formatter</b><br>** Structs<br>** Pipeline: Step API<br>** Token Macro<br><b>Build Timeout</b> |
| ⌚ Timestampper   | ⌚ Workspace Cleanup             | ⌚ Ant                               | ⌚ Gradle               |   |
| ⌚ Pipeline       | ⌚ GitHub Branch Source          | ⌚ Pipeline: GitHub Groovy Libraries | ⌚ Pipeline: Stage View |   |
| ⌚ Git            | ⌚ SSH Build Agents              | ⌚ Matrix Authorization Strategy     | ⌚ PAM Authentication   |   |
| ⌚ LDAP           | ⌚ Email Extension               | ⌚ Mailer                            |                        |   |

\*\* - required dependency

Getting Started

Create First Admin User

계정명:

암호:

암호 확인:

이름:

이메일 주소:

Jenkins 2.319.1

Skip and continue as admin

Save and Continue

설치가 완료되면 계정의 정보 입력

## 4. Jenkins 파이프라인 생성

Jenkins

검색

JeonWoonKi

로그아웃

Dashboard

새로운 Item

사람

빌드 기록

Jenkins 관리

My Views

Lockable Resources

New View

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중

2 대기 중

Jenkins에 오신 것을 환영합니다.

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job

Set up a distributed build

Set up an agent







Configure a cloud

Learn more about distributed builds

## 필요 플러그인 설치

- Generic Webhook Trigger Plugin
- GitLab
- Gitlab API Plugin
- GitLab Authentication plugin
- Mattermost Notification Plugin
- Publish Over SSH

#### Credentials

| T   | P   | Store ↓ | Domain   | ID       | Name                               |
|---|---|---------|----------|----------|------------------------------------|
|  |  | System  | (global) | GitLabID | ts7681@naver.com/***** (깃랩 설정)     |
|  |  | System  | (global) | Blockus  | GitLab API token (Blockus pj)      |
|  |  | System  | (global) | lyy      | dldbud112@naver.com/***** (lyy입니다) |

Jenkins 관리 → Manage Credentials → Credentials 등록

## Item 생성(Pipeline)

## Build Triggers

- ☐ Build after other projects are built ?
- ☐ Build periodically ?
- ☒ Build when a change is pushed to GitLab. GitLab webhook URL: `http://j8a209.p.ssafy.io:8080/project/donjoBackend` ?

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

- ☒ Approved Merge Requests (EE-only)
- ☒ Comments

Comment (regex) for triggering a build ?

Jenkins please retry a build

고급 ▾

- ☐ Generic Webhook Trigger ?
- ☐ GitHub hook trigger for GITScm polling ?

저장

Apply

Definition

Pipeline script from SCM

SCM ?

Git

Repositories ?

Repository URL ?

https://lab.ssfy.com/s08-blockchain-contract-sub2/G08P22A209

Credentials ?

dlbduud112@naver.com/\*\*\*\*\* (이ջ입니다)

Add +

Repository browser ?

(자동)

Additional Behaviours

Add +

Script Path ?

jenkins/backjenkinsfile

☒ Lightweight checkout ?


Pipeline Syntax


저장 Apply

설정 셋팅하기

## 5. Webhook 연결(Build Triggers URL과 Secret token 복사)





 S08P22A209


 Project information


 Repository


 Issues 0


 Merge requests 0


 CI/CD


 Security & Compliance


 Deployments


 Packages and registries

 Infrastructure

 Monitor

 Analytics

 Wiki

 Snippets

 Settings

General

Integrations

**Webhooks**

Access Tokens

Repository

Merge requests

CI/CD

## 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.

### Secret token

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

### Trigger

☒ Push events

Push to the repository.

GitLab → Settings → WebURL Secret token 복사 붙여넣기

## 6. 프로젝트 이미지 생성 후 컨테이너 띄우기

### Backend Pipeline

```
pipeline{
  agent any
  stages{
    stage('백엔드 자동 배포') {
      stages {
        stage('gradlew 권한'){
          steps{
            dir('backend'){
              sh "chmod +x gradlew"
            }
          }
        }
        stage('백엔드 이미지 생성'){
          steps{
            dir('backend'){
              sh "../gradlew bootBuildImage"
            }
          }
        }
        stage('백엔드 컨테이너 삭제'){
          steps{
            catchError{
              sh "docker rm --force backend"
            }
          }
        }
        stage('백엔드 컨테이너 생성') {
          steps {
            sh "docker run -d -p 8081:8080 --name backend backend:0.0.1-SNAPSHOT"
          }
        }
      }
    }
  }
}
```

## Frontend Pipeline

```
pipeline {
    agent any

    stages{
        stage('프론트엔드 자동 배포'){

            stages{
                stage('프론트엔드 이미지 생성'){
                    steps{
                        dir('frontend')
                        {
                            dir('don-jo-app'){
                                sh "docker build -t react-image ."
                            }
                        }
                    }
                }

                stage('프론트엔드 컨테이너 삭제'){
                    steps{
                        catchError{
                            sh "docker rm --force frontend"
                        }
                    }
                }

                stage('컨테이너 생성') {
                    steps {
                        sh "docker run -d -p 3000:3000 --name frontend react-image"
                    }
                }
            }
        }
    }
}
```

## Frontend Dockerfile

```
# 가져올 이미지를 정의
FROM node:14
# 경로 설정하기
WORKDIR /app
# package.json 워킹 디렉토리에 복사 (.은 설정한 워킹 디렉토리를 뜻함)
COPY package.json .
# 명령어 실행 (의존성 설치)
RUN npm install
# 현재 디렉토리의 모든 파일을 도커 컨테이너의 워킹 디렉토리에 복사한다.
COPY . .

# 각각의 명령어들은 한줄 한줄씩 캐싱되어 실행된다.
# package.json의 내용은 자주 바뀌진 않을 거지만
# 소스 코드는 자주 바뀌는데
# npm install과 COPY . . 를 동시에 수행하면
# 소스 코드가 조금 달라질때도 항상 npm install을 수행해서 리소스가 낭비된다.
```

```
# 3000번 포트 노출
EXPOSE 3000

# npm start 스크립트 실행
CMD ["npm", "start"]

# 그리고 Dockerfile로 docker 이미지를 빌드해야한다.
# $ docker build .
```

## Guide Pipeline

```
pipeline{
  agent any
  stages{
    stage('가이드 자동 배포'){

      stages{
        stage('가이드 이미지 생성'){
          steps{
            dir('frontend-guide')
            {
              dir('my-website'){
                sh "docker build -t guide-image2 ."
              }
            }
          }
        }
        stage('가이드 컨테이너 삭제'){
          steps{
            catchError{
              sh "docker rm --force guide"
            }
          }
        }

        stage('컨테이너 생성') {
          steps {

            sh "docker run -d -p 3100:80 --name guide guide-image2"

          }
        }
      }
    }
  }
}
```

## Guide Dockerfile

```
## Base #####
# Use a larger node image to do the build for native deps (e.g., gcc, python)
FROM node:lts as base

# Reduce npm log spam and colour during install within Docker
ENV NPM_CONFIG_LOGLEVEL=warn
ENV NPM_CONFIG_COLOR=false

# We'll run the app as the `node` user, so put it in their home directory
```

```

WORKDIR /home/node/app
# Copy the source code over
COPY --chown=node:node . /home/node/app/

## Development #####
# Define a development target that installs devDeps and runs in dev mode
FROM base as development
WORKDIR /home/node/app
# Install (not ci) with dependencies, and for Linux vs. Linux Musl (which we use for -alpine)
RUN npm install
# Switch to the node user vs. root
USER node
# Expose port 3000
EXPOSE 80
# Start the app in debug mode so we can attach the debugger
CMD ["npm", "start"]

## Production #####
# Also define a production target which doesn't use devDeps
FROM base as production
WORKDIR /home/node/app
COPY --chown=node:node --from=development /home/node/app/node_modules /home/node/app/node_modules
# Build the Docusaurus app
RUN npm run build

## Deploy #####

## Deploy #####
# Use a stable nginx image
FROM nginx:stable-alpine as deploy
WORKDIR /home/node/app
# Copy what we've installed/built from production
COPY --chown=node:node --from=production /home/node/app/build /usr/share/nginx/html/
# Add custom Nginx configuration
COPY --chown=node:node nginx.conf /etc/nginx/conf.d/default.conf

```

All +

| S | W | Name ↓        | 최근 성공          | 최근 실패            | 최근 소요 시간     |   |
|---|---|---------------|----------------|------------------|--------------|---|
| ✓ | ☀ | donjoBackend  | 2 hr 7 min #73 | 3 days 23 hr #49 | 45 sec       | ▶ |
| ✓ | ☀ | donjoFrontend | 2 hr 7 min #70 | 11 days #1       | 5.3 sec      | ▶ |
| ✓ | ☀ | donjoGuide    | 2 hr 7 min #67 | 7 days 22 hr #15 | 1 min 36 sec | ▶ |

3개의 Pipeline 생성

## 7. MYSQL 생성

```

// MySQL Docker 이미지 다운로드
docker pull mysql
// MySQL Docker 컨테이너 생성 및 실행
docker run --name mysql-container -e MYSQL_ROOT_PASSWORD=<password> -d -p 3306:3306 mysql:latest
// Docker 컨테이너 리스트 출력
docker ps -a

```

## 8. Nginx 설정

```
server {
    listen 80;
    server_name j8a209.p.ssafy.io;
    return 301 https://j8a209.p.ssafy.io$request_uri;
}

server {
    listen 443 ssl http2;
    server_name j8a209.p.ssafy.io;

    ssl_certificate /etc/letsencrypt/live/j8a209.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/j8a209.p.ssafy.io/privkey.pem;

    location / {
        proxy_pass http://j8a209.p.ssafy.io:3000;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
    location ~* ^/(api|swagger-ui|swagger-resources|v2/api-docs|v3/api-docs|webjars) {
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header Host $host;
        proxy_pass http://127.0.0.1:8081;
    }

    location /guides {
        proxy_pass http://j8a209.p.ssafy.io:3100;
        proxy_set_header Host $http_host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
        proxy_set_header X-Forwarded-Proto $scheme;
    }
}

server {
    if ($host = j8a209.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
    listen 80;
    server_name example.com;
    return 404; # managed by Certbot
}
```

## 9. Blockchain 엔드포인트 설정





Infura를 통해 폴리곤 체인의 엔드포인트를 발급 받습니다.

app.infura.io/dashboard/ethereum/ac3a17c914fd47a29cb5ed54315f746a/settings/endpoints

### Endpoints

Our Web3 API Key works across several networks, use it on one or use it on all.

**Https** WebSockets REFINE

|  |         |  |                 |
|--|---------|--|-----------------|
|  Ethereum | mainnet | https://mainnet.infura.io/v3/          |                 |
|  Polygon  | mainnet | https://polygon-mainnet.infura.io/v3/  |                 |
|  Optimism | mainnet | https://optimism-mainnet.infura.io/v3/ |                 |
|  Arbitrum |         |  | ACTIVATE ADD ON |

Copy and paste the code below into terminal and your response should be the current block number.

```
curl --url https://mainnet.infura.io/v3/ac3a17c914fd47a29cb5ed54315f746a \
-X POST \
-H "Content-Type: application/json" \
-d '{"jsonrpc": "2.0", "method": "eth_blockNumber", "id": 1}'
```

Your response should look like:

```
{ "jsonrpc": "2.0", "id": 1, "result": "0x8d8e" }
```

☐ PROJECT SECURITY