포팅 매뉴얼

태그	산출물
날짜	

▼ 목차

1. 개발 환경

소통채널

- Mattermost
- Discord
- Notion
- FigJam

데이터베이스

- MariaDB : 10.11.2

서버

- AWS EC2
- Ubuntu : 20.04 LTS
- Docker : 23.0.4

Raspberry Pi

- Raspbian : GNU/Linux 11

이슈관리

- Jira

UI/UX

- Figma

형상관리

- GitLab

IDE

- IntelliJ IDEA : 2022.3.1
- Visual Studio Code : 1.74.2

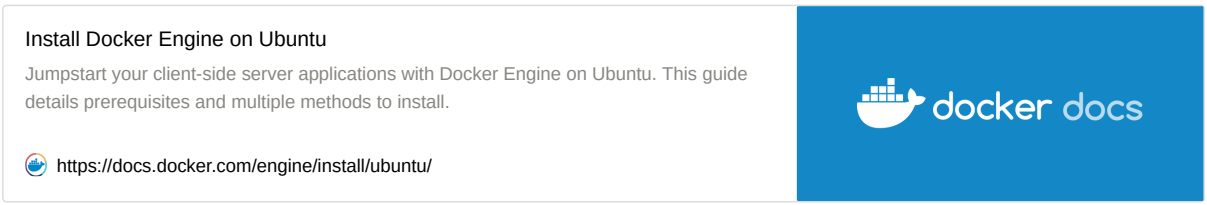
기타 툴

- Postman : 10.13.5
- Swagger :
- Jenkins

2. 환경 세팅

1) Docker 설치

- Ubuntu 20.04 기준, 아래의 가이드를 따라 Docker를 설치한다.



2) MariaDB 설치 및 설정

1. MariaDB 이미지를 내려받는다.

```
docker pull mariadb
```

2. 컨테이너 실행

```
docker run -p 3206:3206 --name mariadb -e MARIADB_ROOT_PASSWORD=
{비밀번호} -d mariadb
```

3. DB 사용자 추가

   1) mariadb 컨테이너 접속

```
docker exec -it mariadb /bin/bash
```

   2) 로그인, 아래의 커맨드를 입력하고 컨테이너를 생성할 때 입력한 비밀번호로 로그인한다.

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

   3) 모든 DB, 테이블에 접속 가능한 사용자 생성

```
use mysql
CREATE USER '{사용자 이름}'@'%' IDENTIFIED BY '{비밀번호}';
GRANT ALL PRIVILEGES ON *.* TO '{사용자 이름}'@'%';
FLUSH PRIVILEGES;
```

3) Nginx 설치 및 설정

1. Nginx, Certbot 이미지를 내려받는다.

```
docker pull nginx:1.15-alpine
docker pull certbot/certbot
```

2. 서버 최상단 폴더에 docker-compose.yml 파일을 생성

```
cd /
sudo vi docker-compose.yml
```

3. docker-compose.yml 파일 설정

```
version: '3.2'
services:
  nginx:
    container_name: nginx
    image: nginx:1.15-alpine
    volumes:
      - ./data/nginx:/etc/nginx/conf.d
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    ports:
      - "80:80"
      - "443:443"
    expose:
      - "80"
    command: "/bin/sh -c 'while :; do sleep 6h & wait $${!}; nginx -s reload; done & nginx -g \"daemon off;\"'"
  certbot:
    container_name: certbot
    image: certbot/certbot
    restart: unless-stopped
    volumes:
      - ./data/certbot/conf:/etc/letsencrypt
      - ./data/certbot/www:/var/www/certbot
    entrypoint: ["/bin/sh", "-c", "trap exit TERM; while :; do certbot renew; sleep 12h & wait $${!}; done;"]
```

4. conf.d 파일 설정

- https 설정
- 백엔드, 프론트엔드 주소 분기

```
server {
  listen 80 default_server;
  server_name {도메인 주소};
  server_tokens off;
  client_max_body_size 1G; # media type 파일 용량 설정

  location /.well-known/acme-challenge/ {
  root /var/www/certbot;
  }

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

server {
  listen 443 ssl;
  server_name {도메인 주소};
  server_tokens off;
  client_max_body_size 1G;

  ssl_certificate /etc/letsencrypt/live/{도메인 주소}/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/{도메인 주소}/privkey.pem;
  include /etc/letsencrypt/options-ssl-nginx.conf;
  ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

  location /api {
  proxy_pass http://{도메인 주소}:8080/api;
  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;
    }

    location /{
    proxy_pass http://{도메인 주소}:3000;
    }

  }
```

---

3. 백엔드 빌드 및 배포

- 환경변수 파일은 보안을 위해 git 원격저장소에 업로드 하지 않음.
- 따라서 application.yml 파일 추가 필요

1) application.yml 파일 작성 및 추가

- 위치: /backend/src/main/resources/application.yml

```
spring:
  profiles:
    active: local

  datasource:
    driver-class-name: org.mariadb.jdbc.Driver
    url: jdbc:mariadb://{도메인}:3206/docar
    username: {사용자}
    password: {암호}

  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true

server:
  port: 8080
  servlet:
    context-path: /api

---

spring:
  config:
    activate:
      on-profile: dev


server:
  port: 18090
  servlet:
    context-path: /api

app:
  uri:
    scheme: https
    domain: k8d101.p.ssafy.io
    port: 443

---

spring:
  config:
    activate:
      on-profile: local

server:
```

```
  port: 8080
  servlet:
    context-path: /api

app:
  uri:
    scheme: http
    domain: localhost
    port: 8080

jwt:
  secretKey: {JWT 시크릿 키}
  expirationTime: 86400000
```

2) Spring Boot 프로젝트 빌드

```
./gradlew clean build
```

3) Docker 이미지 생성

```
docker build -t d101-server ./backend
```

4) 배포

- 이미 실행되고 있는 d101-server 컨테이너가 있다면 삭제한다.

```
docker rm -f d101-server
```

- 컨테이너를 생성한다.

```
docker run -d -p 8080:8080 --name d101-server d101-server
```

- 사용한 옵션
  - `-d` :
    - 컨테이너를 백그라운드에서 실행
  - `-p 8080:8080`:
    - 호스트 머신의 `8080`포트와 컨테이너 내부의 `8080` 포트를 매핑
  - `--name d101-server`:
    - 컨테이너 이름을 `d101-server`로 설정

추가) 백엔드 배포 관련 설정 파일 설명

- Dockerfile

```
FROM adoptopenjdk/openjdk11
WORKDIR /usr/app
COPY build/libs/docar-0.0.1-SNAPSHOT.jar d101.jar
EXPOSE 8080
CMD ["java", "-jar", "d101.jar"]
```

4. 프론트엔드 빌드 및 배포

1) Docker 이미지 생성

```
docker build -t d101-client ./frontend
```

2) 배포

- 이미 실행되고 있는 d101-client 컨테이너가 있다면 삭제한다.

```
docker rm -f d101-client
```

- 컨테이너를 생성한다.

```
docker run -d -p 3000:3000 --name d101-client d101-client
```

- 사용한 옵션
  - `-d`:
    - 컨테이너를 백그라운드에서 실행
  - `-p 3000:3000`:
    - 호스트 머신의 `3080` 포트와 컨테이너 내부의 `3000` 포트 매핑
  - `-name d101-clinet`:
    - 컨테이너 이름을 `d101-client`로 설정

추가) 프론트엔드 배포관련 설정 파일 설명

- **Dockerfile**

```
FROM node:lts-alpine as build-stage
WORKDIR /app
COPY package*.json ./
RUN npm install --force
COPY . .
RUN npm run build

FROM nginx:stable-alpine as production-stage
COPY --from=build-stage /app/build /usr/share/nginx/html
COPY nginx.conf /etc/nginx/conf.d/default.conf
EXPOSE 3000
CMD ["nginx", "-g", "daemon off;"]
```

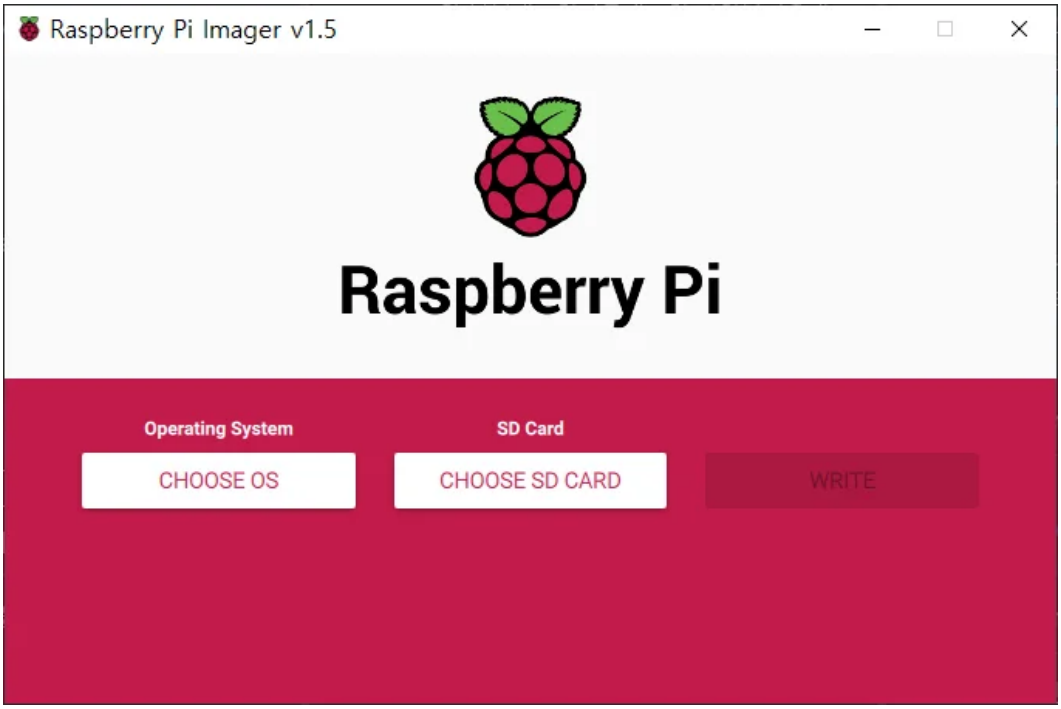
- **nginx.conf**

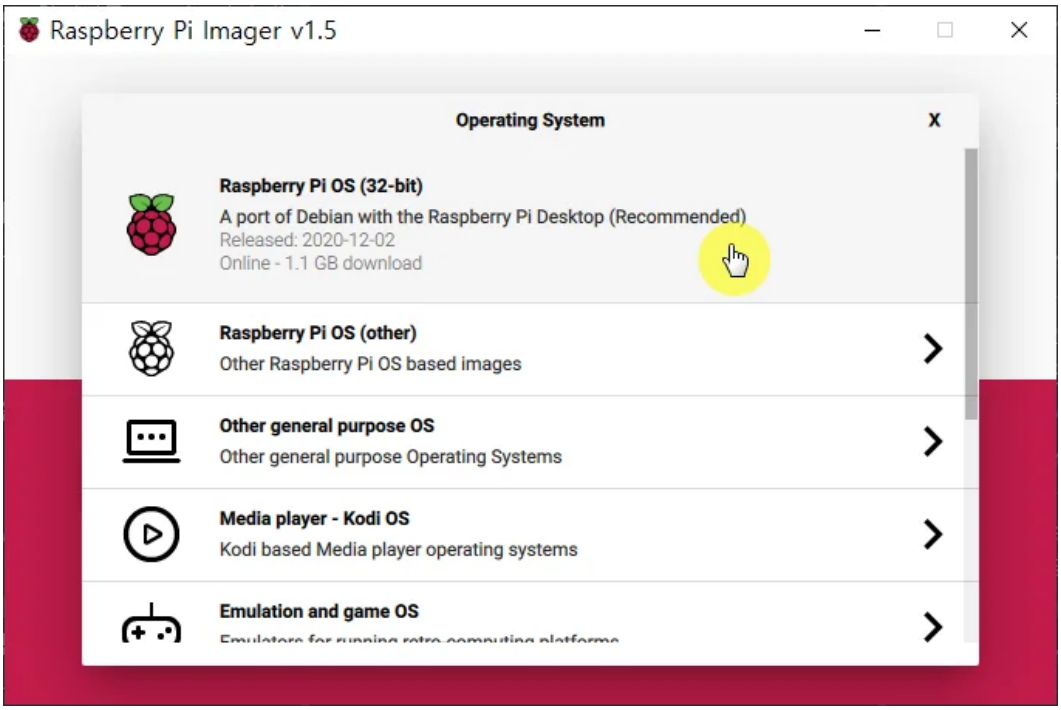
```
server {
  listen       3000;
```

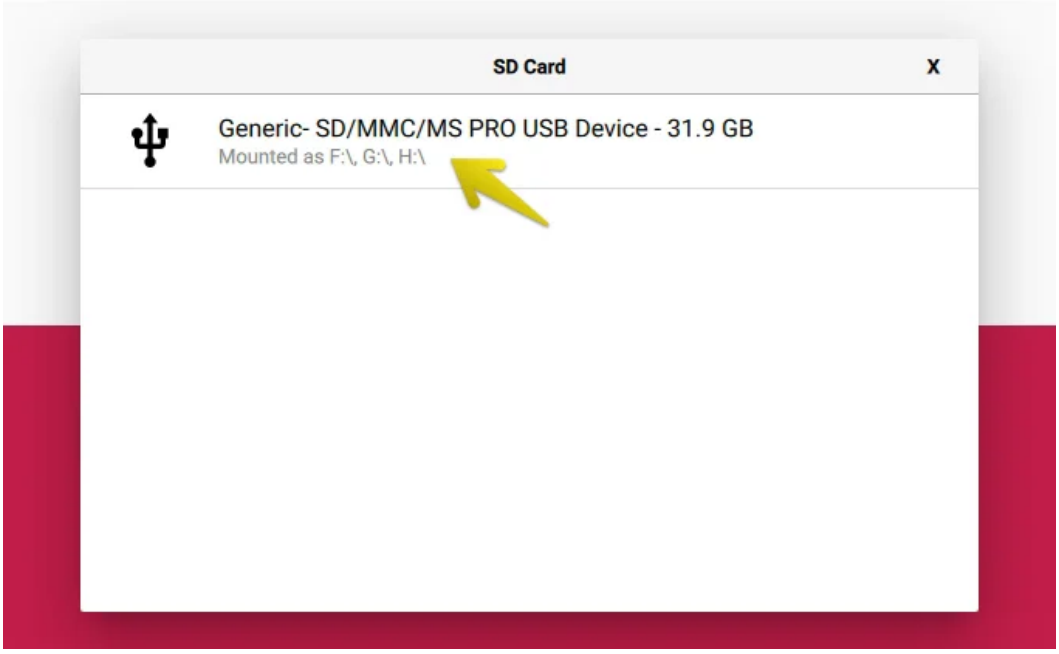
```
    server_name  localhost;
    location / {
      root   /usr/share/nginx/html;
      index  index.html;
      try_files $uri $uri/ /index.html;
    }
}
```

5. IoT 구성

1) Raspberry Pi(카트)







2) TurtleBot

- TurtleBot은 Raspberry Pi, OpenCR, Remote PC로 구성된다.

1) Raspberry Pi

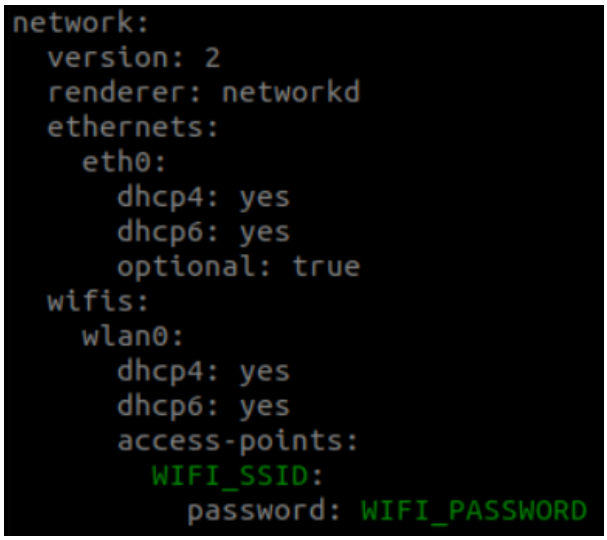
```
Configure the WiFi Network Setting
$ cd /media/$USER/writable/etc/netplan
$ sudo nano 50-cloud-init.yaml

ROS2 Network Configuration
```

```
ROS_DOMAIN_ID=30 #TURTLEBOT3

Install the LDS-02 driver and update TurtleBot3 package
$ sudo apt update
$ sudo apt install libudev-dev
$ cd ~/turtlebot3_ws/src
$ git clone -b ros2-devel https://github.com/ROBOTIS-GIT/ld08_driver.git
$ cd ~/turtlebot3_ws/src/turtlebot3 && git pull
$ rm -r turtlebot3_cartographer turtlebot3_navigation2
$ cd ~/turtlebot3_ws && colcon build --symlink-install

Export the LDS_MODEL to the bashrc file. Depending on your LDS model, use LDS-01 or LDS-02
$ echo 'export LDS_MODEL=LDS-02' >> ~/.bashrc
$ source ~/.bashrc
```



2) OpenCR

```
$ sudo dpkg --add-architecture armhf
$ sudo apt update
$ sudo apt install libc6:armhf

$ export OPENCR_PORT=/dev/ttyACM0
$ export OPENCR_MODEL=burger
$ rm -rf ./opencr_update.tar.bz2

$ wget https://github.com/ROBOTIS-GIT/OpenCR-Binaries/raw/master/turtlebot3/ROS2/latest/opencr_update.tar.bz2
$ tar -xvf ./opencr_update.tar.bz2

$ cd ~/opencr_update
$ ./update.sh $OPENCR_PORT $OPENCR_MODEL.opencr
```

3) Remote PC

```
Install ROS 2 on Remote PC
$ wget https://raw.githubusercontent.com/ROBOTIS-GIT/robotis_tools/master/install_ros2_foxy.sh
$ sudo chmod 755 ./install_ros2_foxy.sh
$ bash ./install_ros2_foxy.sh

Install Gazebo11
$ sudo apt-get install ros-foxy-gazebo-*

Install Cartographer
$ sudo apt install ros-foxy-cartographer
$ sudo apt install ros-foxy-cartographer-ros
```

```
Install Navigation2
$ sudo apt install ros-foxy-navigation2
$ sudo apt install ros-foxy-nav2-bringup

Install TurtleBot3 Packages
$ source ~/.bashrc
$ sudo apt install ros-foxy-dynamixel-sdk
$ sudo apt install ros-foxy-turtlebot3-msgs
$ sudo apt install ros-foxy-turtlebot3

Environment Configuration
$ echo 'export ROS_DOMAIN_ID=30 #TURTLEBOT3' >> ~/.bashrc
$ source ~/.bashrc
```