포팅메뉴얼

담당자	Jeong-b 근창 박
상태	진행 중
태그	

C203 서버 포팅메뉴얼

1. EC2 서버 기본 설정

- 서버 시간을 한국어 표준시로 변경

```
sudo timedatectl set-timezone Asia/Seoul
```

- 미러 서버를 카카오 서버로 변경

```
 sudo vi /etc/apt/sources.list

# 해당 파일 내 kr.archive.ubuntu.com를 mirror.kakao.com로 변경
:%s/ap-northeast-2.ec2.archive.ubuntu.com/mirror.kakao.com/
:wq
```

- 패키지 목록 업데이트 및 패키지 업데이트

```
sudo apt update
sudo apt upgrade
```

- 가상메모리 증설 (AWS 참고사이트)

```
# 현재 메모리 확인
free -h

# 16G(256M*64)의 가상메모리 증설
# bs(블록크기), count(블록개수)의 곱이 총 메모리 크기
sudo dd if=/dev/zero of=/swapfile bs=256M count=64

# 스왑 파일으리 읽기 쓰기 권한 엡데이트
sudo chmod 600 /swapfile

# Linux 스왑 영역 설정
sudo mkswap /swapfile

# 스왑 파일 즉시 사용
sudo swapon /swapfile

# 스왑 프로세스 확인
sudo swapon -s

# 부팅시 스왑파일 적용
sudo vi /etc/fstab

# 해당 파일 마지막에 작성 후 저장
/swapfile swap swap defaults 0 0
```

- 방화벽 설정

```
# 방화벽 상태 조회
sudo ufw status

# 방화벽 설정(80 - http, 443 - https)
sudo ufw allow 80
sudo ufw allow 443
```

2. NGINX & SSL 설정

- Nginx 설치

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

# 2. Niginx 상태확인 - active면 정상
sudo systemctl status nginx

# 3. Nginx 환경설정
sudo vi /etc/nginx/site-available/{파일명}.conf
```

- SSL 설정

```
# 1. let's Encrypt 설치
sudo apt-get install letsencrypt -y

# 2. Certbot 설치
sudo apt-get install certbot python3-certbot-nginx

# 3. Ceartbot 동작
sudo certbot --nginx
  # 이메일 입력
  # 약관 동의 : A
  # 이베일 수신 동의 : Y
  # 도메인 입력 : k9c203.p.ssafy.io
  # http 입력시 리다익트 여부 : 2
```

3. Docker

```
# https를 통해 레포지토리를 사용할 수 있도록하는 패키지 설치
sudo apt install ca-certificates curl gnupg

# Docker의 공식 GPG 키를 추가
sudo install -m 0755 -d /etc/apt/keyrings
curl -fsSL https://download.docker.com/linux/ubuntu/gpg | sudo gpg --dearmor -o /etc/apt/keyrings/docker.gpg
sudo chmod a+r /etc/apt/keyrings/docker.gpg

# 리포지토리를 설정
echo \
"deb [arch="$(dpkg --print-architecture)" signed-by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/ubuntu \
"$(. /etc/os-release && echo "$VERSION_CODENAME")" stable" | \
sudo tee /etc/apt/sources.list.d/docker.list > /dev/null

# 패키지 index를 업데이트
sudo apt-get update

# Docker Engine, containerd 및 Docker Compose를 설치
sudo apt-get install docker-ce docker-ce-cli containerd.io docker-buildx-plugin docker-compose-plugin

# docker 설치 확인
docker -v
```

4. MariaDB (Docker) - Docker Hub 공식 사이트

```
# mariaDB의 이미지를 pull한다.
sudo docker pull mariadb

# 가져온 이미지를 컨테이너로 실행한다
sudo docker run -d -p 13306:3306 -v /var/lib/mysql:/var/lib/mysql -e MYSQL_ROOT_PASSWORD={root계정비밀번호} -e MARIADB_USER={user이름} -e M

# mariaDB 접속
sudo docker exec -it mariaDB /bin/bash
mariadb -u root -p
비밀번호입력

# mariaDB에 database 생성
CREATE DATABASE {데이터베이스이름}

# user 권한 부여
GRANT ALL PRIVILEGES ON {데이터베이스이름}.* TO  'user명'@'%'

# 변경된 권한을 부여
FLUSH PRIVILEGES;
```

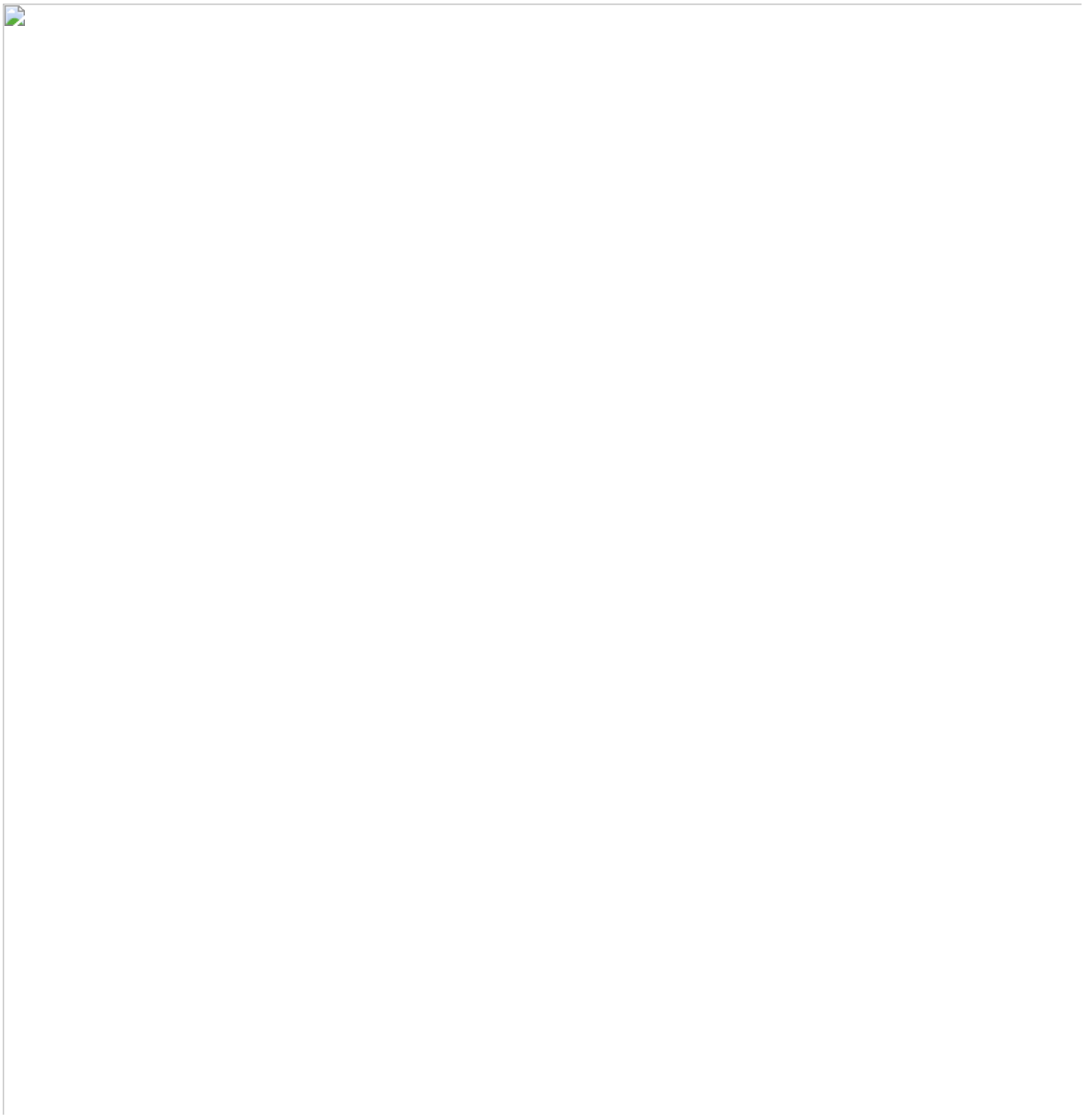
5. Redis - Redis 공식 사이트

```
# Redis에 필요한 패키지를 먼저 설치한다.
sudo apt install lsb-release curl gpg

# Redis를 설치한다.
curl -fsSL https://packages.redis.io/gpg | sudo gpg --dearmor -o /usr/share/keyrings/redis-archive-keyring.gpg

echo "deb [signed-by=/usr/share/keyrings/redis-archive-keyring.gpg] https://packages.redis.io/deb $(lsb_release -cs) main" | sudo tee /

sudo apt-get update
sudo apt-get install redis

# Redis의 설정을 변경하기 위해 redis.conf의 권한을 수정한다.
sudo chown root:root /etc/redis/redis.conf

# Redis의 설정을 변경한다.
sudo vi /etc/redis/redis.conf
  # requirepass 비밀번호
  # bind 접근 가능한 ip
  # port 포트번호

# 다시 권한을 원래대로 돌린다.
sudo chown redis:redis /etc/redis/redis.conf

# 바뀐 설정을 적용시킨다.
sudo systemctl restart redis-server.service

# 재부팅시에도 자동으로 실행되되록 한다.
sudo systemctl enable redis-server.service
```

6. Jenkins (Docker)

```
# 1. Jenkins 이미지 파일을 JDK 버전에 맞춰 pull한다.
sudo docker pull jenkins/jenkins:lts-jdk11

# 2. 가져온 이미지를 컨테이너로 올린다.
sudo docker run -d -p 8888:8080 -v /home/ubuntu/jenkins:/var/jenkins_home -v /usr/bin/docker:/usr/bin/docker -v /var/run/docker.sock:/v

# 3. jenkins WEB (k9c203.p.ssafy.io:8888)에 접속한 후 초기 비밀번호를 입력한다.
sudo docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword

# 4. plugin을 설치한다.

# 5. jenkins 로그인 설정을 한다.
```

3. jenkins WEB (k9c203.p.ssafy.io:8888)에 접속한 후 초기 비밀번호를 입력한다.



4. plugin을 설치한다.

5. jenkins 로그인 설정을 한다.

- 시간대 설정
  - *사용자 → 설정 → User Defined Time Zone*
    - Asia/Seoul 설정

- GitLab 등록
  1. Settings → Access Tokens 접속
  2. Token Name 입력 Expiration date 필요한 경우 입력
  3. select a role ⇒ Maintainer 로 설정, select scopes ⇒ 모든 항목 다 선택
  4. Create project access token 하여 토큰 생성(**토큰은 1회만 생성되므로 복사하여 저장할것**)

- 플러그인 설치
  - *Jenkins 관리 → Plugins → Available plugins*
    - Deploy to container

    - Post build task
    - NodeJS
    - GitLab
    - Mattermost Notification

- Credentiasl 추가
  - *Jenkins 관리 → Credentials → System → Global credentials (unrestricted) → Add Credentials*
    - 소스코드 관리용 Credentiasl
      - Kind → Username with password선택 후,
        Username을 깃랩 ID, Password를 Acess tocken,
      - ID, Description 적절히 입력 후 create 눌러 생성하기
      - Scope는 Global로 설정
    - build triggger용 Credentiasl
      - Kind → GitLab API token 선택 후, API token에 위에서 생성한 access token 입력
      - ID, Description 적절히 입력 후 create 눌러 생성하기
      - Scope는 Global로 설정

- GitLab 연결
  - *System → Gitlab*
    - connection 이름 적절히 설정
    - GitLab host URL은 “https://lab.ssafy.com”로 설정
    - Credentials은 위에서 생성한 credentials로 선택 후 ⇒ Test Connection
    - 정상적으로 되었는지 확인 ⇒ “Apply” → “저장”

- Mattermost 연결
  - *System → Global Mattermost Notifier Settings*
    - **Endpoint**: mattermost webhook 입력
    - **Channel**: Incoming Webhook을 추가할 때 설정했던 채널 이름
    - **Build Server URL**: Jenkins 주소 (자동으로 입력되어 있을 것이다.)

- Manage Jenkins
  - *System →Global Properties*
    - Environetn variables 체크
    - CI, false 환경변수 추가
    - Save 클릭

- NodeJS 설정
  - Jenkins 관리 →Tools → **NodeJS installations**
    - name : nodejs 설정
    - version : PJT에 맞게 설정

8. Nginx.conf 수정

- EC2 Nginx.conf
  /etc/nginx/sites-available/default

```
server {
        server_name {도메인명};

        location / {
                proxy_pass http://localhost:{프론트 포트번호};
        }

  location /api/ {
                proxy_pass http://localhost:{백엔드 포트번호};
        }

  proxy_set_header X-Real-IP $remote_addr;
  proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_for;
  proxy_set_header Host $http_host;
  proxy_set_header X-Forwarded-Proto $scheme;

      listen [::]:443 ssl ipv6only=on;
      listen 443 ssl;
      ssl_certificate /etc/letsencrypt/live/{도메인명}/fullchain.pem;
      ssl_certificate_key /etc/letsencrypt/live/{도메인명}/privkey.pem;
      include /etc/letsencrypt/options-ssl-nginx.conf;
      ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;
}
server {
   if ($host = {도메인명}) {
       return 301 https://$host$request_uri;
    }
        listen 80 ;
        listen [::]:80 ;
    server_name {도메인명};
      return 404;
}
server {
        listen {포트번호} ssl;
        server_name {도메인명};

        ssl_certificate /etc/letsencrypt/live/{도메인명}/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/{도메인명}/privkey.pem;

        location / {
                proxy_pass http://localhost:{젠킨스 포트번호};
        }
}
server {
  listen {포트번호} ssl;
  server_name {도메인명};

  ssl_certificate /etc/letsencrypt/live/{도메인명}/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/{도메인명}/privkey.pem;

  location / {
    proxy_pass http://localhost:{웹소켓 포트번호};
    proxy_http_version 1.1;
          proxy_set_header Upgrade $http_upgrade;
          proxy_set_header Connection "upgrade";
  }
}
server {
        listen {포트번호} ssl;
        server_name {도메인명};

        ssl_certificate /etc/letsencrypt/live/{도메인명}/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/{도메인명}/privkey.pem;

        location / {
                proxy_pass http://localhost:{웹소켓 포트번호};
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
                proxy_connect_timeout 300;
                proxy_send_timeout 300;
                proxy_read_timeout 300;
                send_timeout 300;
        }
}
server {
        listen {포트번호} ssl;
        server_name {도메인명};
```

```
        ssl_certificate /etc/letsencrypt/live/{도메인명}/fullchain.pem;
        ssl_certificate_key /etc/letsencrypt/live/{도메인명}/privkey.pem;

        location / {
                proxy_pass http://localhost:{웹소켓 포트번호};
                proxy_http_version 1.1;
                proxy_set_header Upgrade $http_upgrade;
                proxy_set_header Connection "upgrade";
        }
}
```

- Client

```
server {
    listen 80;
    server_name localhost;

    location / {
        root /app/build;
        index index.html;
        try_files $uri $uri/ /index.html;
    }
}
```

9. DockerFile

- Client

```
# nginx 이미지를 사용합니다. 뒤에 tag가 없으면 latest 를 사용합니다.
FROM nginx

# work dir 고정
WORKDIR /app

# work dir 에 build 폴더 생성 /app/build
RUN mkdir ./build

# host pc의 현재경로의 build 폴더를 workdir 의 build 폴더로 복사
ADD ./build ./build

# nginx 의 default.conf 를 삭제
RUN rm /etc/nginx/conf.d/default.conf

# host pc 의 nginx.conf 를 아래 경로에 복사
COPY ./nginx.conf /etc/nginx/conf.d

# 포트 오픈
EXPOSE 3000
EXPOSE 80
EXPOSE 443
```

- Server

```
# 기본 이미지로 OpenJDK 11을 사용
FROM openjdk:11-jre-slim
WORKDIR /app
COPY .env /app/.env

# JAR 파일을 컨테이너에 복사
COPY build/libs/server-0.0.1-SNAPSHOT.jar /app/server-0.0.1-SNAPSHOT.jar

# 외부와 통신할 포트 지정
EXPOSE 8080
EXPOSE 18080

# 컨테이너가 시작될 때 실행할 명령 지정
CMD ["java", "-jar", "/app/server-0.0.1-SNAPSHOT.jar"]
```

- socket-HW

```
FROM python:3.9

WORKDIR /app
```

```
COPY . /app

RUN apt-get update && apt-get install -y python3-pip

RUN pip install -r requirements.txt

EXPOSE 12345

CMD ["python3", "server.py"]
```

- websocket-HW

```
FROM python:3.9

WORKDIR /app

COPY . /app

RUN apt-get update && apt-get install -y python3-pip

RUN pip install -r requirements.txt

EXPOSE 18090

CMD ["python3", "server.py"]
```

- websocket_web_app

```
# node 이미지를 사용합니다.
FROM node:18.18.2

# work dir 고정
WORKDIR /server

# host pc의 현재경로의 폴더를 workdir의 폴더로 복사
ADD ./ ./

# 5000 포트 오픈
EXPOSE 7777

# container 실행 시 자동으로 실행할 command. node 시작함
CMD ["node", "back-websocket.js"]
```

- websocket_web_hw

```
# node 이미지를 사용합니다.
FROM node:18.18.2

# work dir 고정
WORKDIR /server

# host pc의 현재경로의 폴더를 workdir의 폴더로 복사
ADD ./ ./

# 5000 포트 오픈
EXPOSE 7776

# container 실행 시 자동으로 실행할 command. node 시작함
CMD ["node", "server.js"]
```

10. Pipeline

```
pipeline {
    agent any
    tools {nodejs "nodejs"}

    stages {
        // before service
        stage('Before service') {
            steps {
                echo 'stop Container'
                sh 'docker stop Client'
```

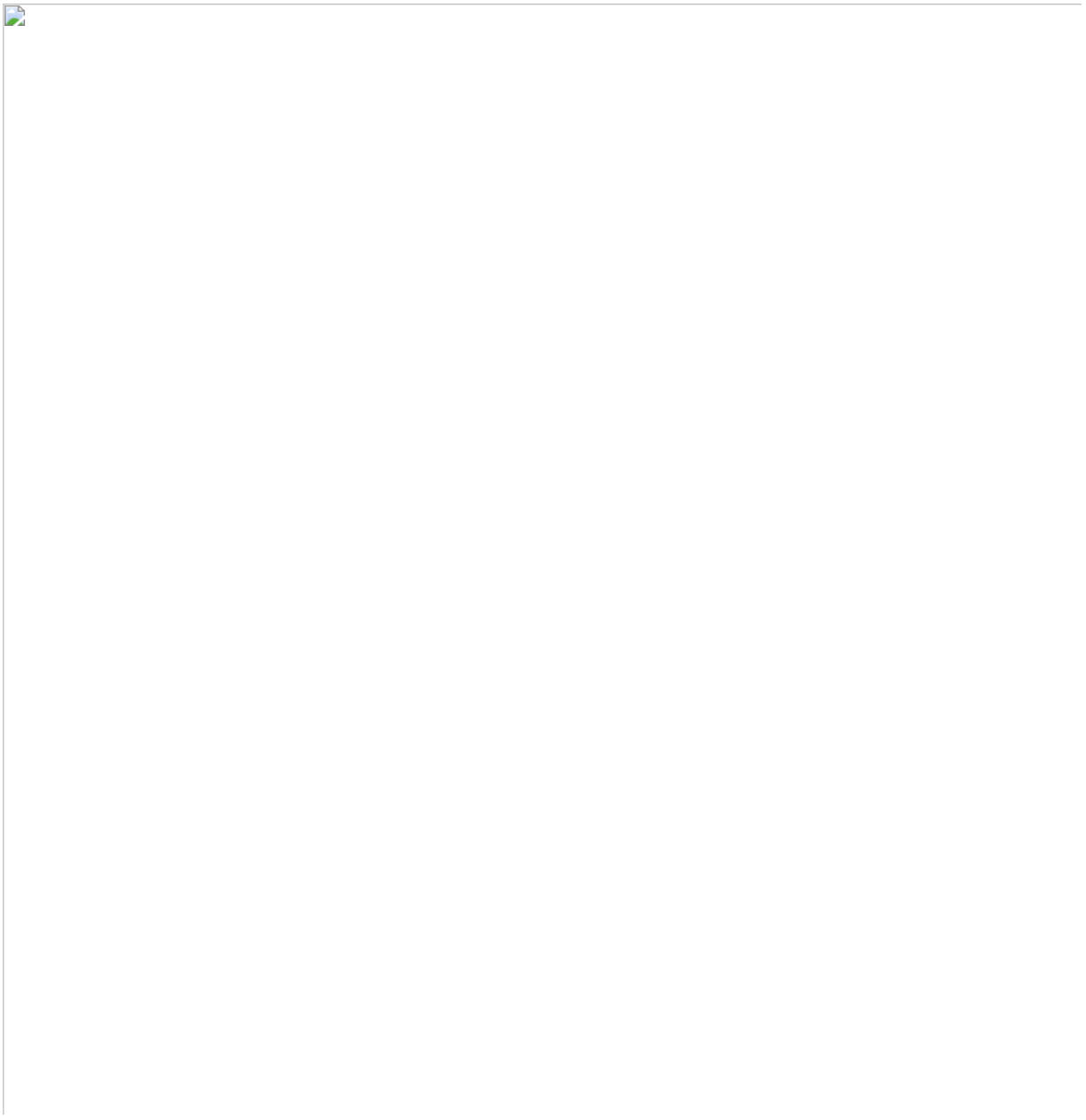
```
                sh 'docker stop Server'
                sh 'docker stop Socket-HW'
                sh 'docker stop Websocket-HW'
                sh 'docker stop Websocket-APP'
                sh 'docker stop Websocket-Web'
                echo 'remove container'
                sh 'docker rm Client'
                sh 'docker rm Server'
                sh 'docker rm Socket-HW'
                sh 'docker rm Websocket-HW'
                sh 'docker rm Websocket-APP'
                sh 'docker rm Websocket-Web'
                echo 'remove Image'
                sh 'docker rmi client'
                sh 'docker rmi server'
                sh 'docker rmi socket-hw'
                sh 'docker rmi websocket-hw'
                sh 'docker rmi websocket-app'
                sh 'docker rmi websocket-web'
            }
        }

        // git clone
        stage('Git Clone') {
            steps {
                echo 'Git Clone'
                git branch: 'develop', credentialsId: 'c203planet', url: 'https://lab.ssafy.com/s09-final/S09P31C203.git'
            }
        }

        // move env
        stage('move env') {
            steps {
                echo 'move env'
                dir('client') {
                    sh 'cp /var/jenkins_home/.client_env ./.env'
                    sh 'chmod 777 ./.env'
                }
                dir('server') {
                    sh 'cp  /var/jenkins_home/.server_env ./.env'
                    sh 'chmod 777 ./.env'
                }
                dir('socket_HW') {
                    sh 'cp /var/jenkins_home/.socket_hw_env ./.env'
                    sh 'chmod 777 ./.env'
                }
                dir('websocket_HW') {
                    sh 'cp /var/jenkins_home/.websocket_hw_env ./.env'
                    sh 'chmod 777 ./.env'
                }
                dir('websocket_web_app') {
                    sh 'cp /var/jenkins_home/.websocket_web_app_env ./.env'
                    sh 'chmod 777 ./.env'
                }
                dir('websocket_web_HW') {
                    sh 'cp /var/jenkins_home/.websocket_web_hw_env ./.env'
                    sh 'chmod 777 ./.env'
                }
            }
        }

        // build
        stage('Build') {
            steps {
                echo 'npm Build'
                dir('client') {
                    sh 'npm install'
                    sh 'npm run build'
                }
                dir('websocket_web_app') {
                    sh 'npm install'
                }
                dir('websocket_web_HW') {
                    sh 'npm install'
                }

                echo 'jar Buld'
                dir('server') {
                    sh 'chmod +x ./gradlew'
                    sh './gradlew clean bootJar'
                }
            }
        }
```

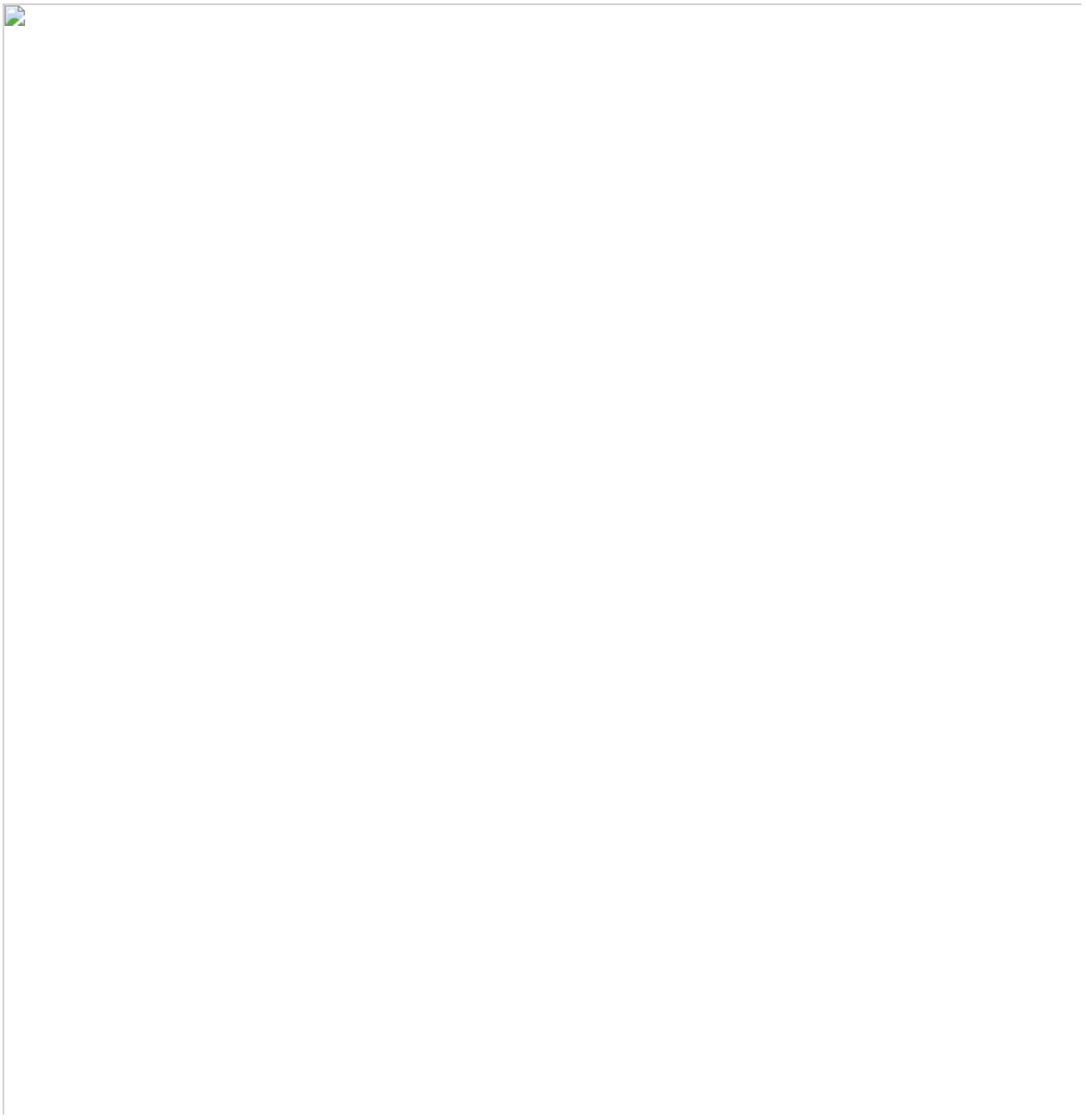
```
        // docker bulid
        stage('Docker Build') {
            steps {
                dir('client') {
                    sh 'docker build -t client .'
                }
                dir('server') {
                    sh 'docker build -t server .'
                }
                dir('socket_HW') {
                    sh 'docker build -t socket-hw .'
                }
                dir('websocket_HW') {
                    sh 'docker build -t websocket-hw .'
                }
                dir('websocket_web_app') {
                    sh 'docker build -t websocket-app .'
                }
                dir('websocket_web_HW') {
                    sh 'docker build -t websocket-web .'
                }
            }
        }

        // docker run
        stage('Docker run') {
            steps {
                sh 'docker run -d -p 3000:80 -e TZ=Asia/Seoul --name Client --network planet client'
                sh 'docker run -d -p 18080:8080 -e TZ=Asia/Seoul -v /home/ubuntu/user:/home/ubuntu/user -e "SPRING_PROFILES_ACTIVE=proc
                sh 'docker run -d -p 12345:12345 -e TZ=Asia/Seoul -v /home/ubuntu/user:/home/ubuntu/user -v /home/ubuntu/character:/hor
                sh 'docker run -d -p 18090:18090 -e TZ=Asia/Seoul -v /home/ubuntu/user:/home/ubuntu/user --name Websocket-HW --network
                sh 'docker run -d -p 7777:7777 -e TZ=Asia/Seoul --name Websocket-APP --network planet websocket-app'
                sh 'docker run -d -p 7776:7776 -e TZ=Asia/Seoul --name Websocket-Web --network planet websocket-web'
            }
        }
    }

    post {
        success {
          script {
                mattermostSend (color: 'good',
                message: "배포 성공   :gmqwhr_:   [:c203_littleplanet:](https://k9c203.p.ssafy.io)",
                )
            }
        }
        failure {
          script {
                mattermostSend (color: 'danger',
                message: "배포 실패   :please1: "
                )
            }
        }
    }
}
```

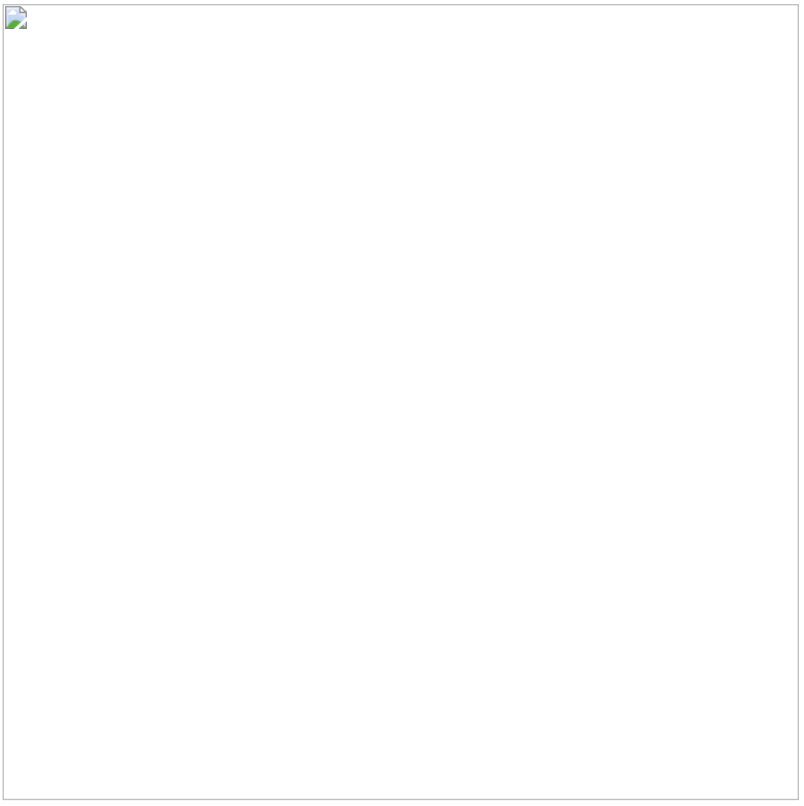
11. React-Native build

- /S09P31C203/LittlePlanet을 로컬로 이동후 해당 폴더에서 아래 명령 실행

```
mkdir ./android/app/src/main/assets
npm i
react-native bundle --platform android --dev false --entry-file index.js --bundle-output ./android/app/src/main/assets/index.android.bu
cd ./android
./gradlew clean // 오류시 아래 내용 실행 후 재실행 하고 다시 아래 실행
//오류가 발생했다면 아래 4번 실행전 build에서 make project 실행 후 4번 진행
```

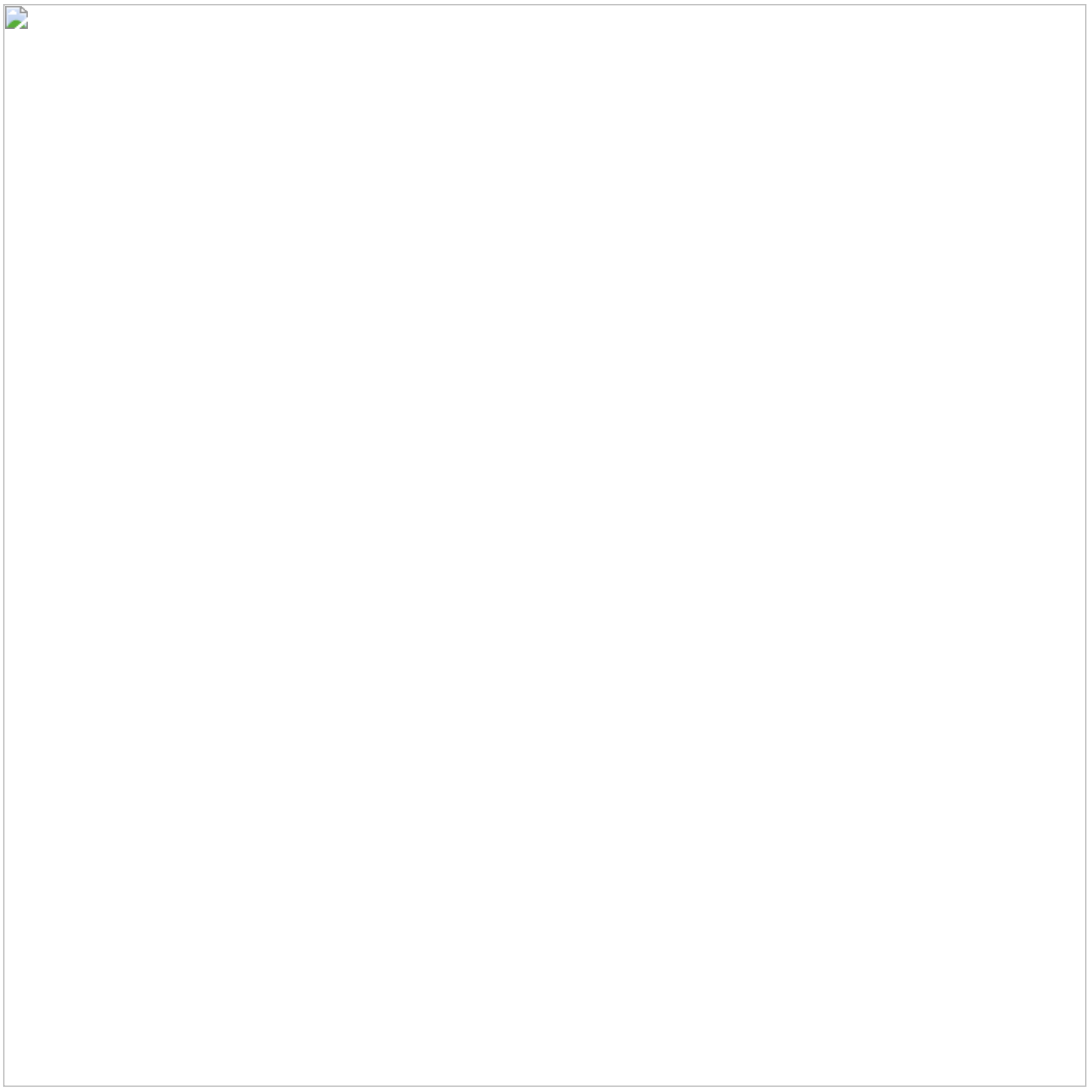






C203 HW 포팅메뉴얼

0. 라즈베리파이 os 다운로드 - 라즈베리파이 이미지 파일



TPU 사용 하기 위해서는 Debian 10버전만 지원하므로 Debian 10버전인 rasbian을 설치해야한다.

1. 기본 설정

- 패키지 목록 업데이트 및 패키지 업데이트

```
sudo apt-get update
sudo apt-get upgrade
sudo reboot
```

- 네트워크 연결

```
# 아래의 명령을 통해 mac주소와 ip주소를 확인한다.
ifconfig

# 설정파일을 변경하기 위해 vim을 설치한다
sudo apt-get install vim

# 네트워크 설정파일을 변경한다.
sudo vi /etc/dhcpcd.conf

# 해당 파일의 마지막 줄에 아래 내용을 추가한다.
interface wlan0
static ip_address={ifconfig ip주소}
static routers={ifconfig ip주소에서 마지막을 1로 변경}

# 저장 후 재실행 한다.
```

```
sudo /etc/init.d/networking restart
sudo reboot
```

- 미러사이트 변경

```
# apt설정파일을 수정한다
sudo vi /etc/apt/sorce.list

# 해당파일의 주소를 수정한다
:%s/raspbian.raspberrypi.org/ftp.kaist.ac.kr\/raspbian
```

- 기기해상도 변경 - 1280 * 720
- 기기오버클럭 설정

```
# 기기 설정파일 접속한다.
sudo vi /boot/config.txt

# 아래 내용으로 변경한다.(bold된 부분을 수정한다.)
# For more options and information see
# http://rpf.io/configtxt
# Some settings may impact device functionality. See link above for details

# uncomment if you get no picture on HDMI for a default "safe" mode
#hdmi_safe=1

# uncomment the following to adjust overscan. Use positive numbers if console
# goes off screen, and negative if there is too much border
#overscan_left=16
#overscan_right=16
#overscan_top=16
#overscan_bottom=16

# uncomment to force a console size. By default it will be display's size minus
# overscan.
#framebuffer_width=1280
#framebuffer_height=720

# uncomment if hdmi display is not detected and composite is being output
hdmi_force_hotplug=1
hdmi_ignore_edid=0xa5000080

# uncomment to force a specific HDMI mode (this will force VGA)
hdmi_group=2
hdmi_mode=4

# uncomment to force a HDMI mode rather than DVI. This can make audio work in
# DMT (computer monitor) modes
hdmi_drive=2

# uncomment to increase signal to HDMI, if you have interference, blanking, or
# no display
#config_hdmi_boost=4

# uncomment for composite PAL
#sdtv_mode=2

#uncomment to overclock the arm. 700 MHz is the default.
arm_freq=2000
over_voltage=6
gpu_freq=700
initial_turbo=30

# Uncomment some or all of these to enable the optional hardware interfaces
#dtparam=i2c_arm=on
#dtparam=i2s=on
#dtparam=spi=on

# Uncomment this to enable infrared communication.
#dtoverlay=gpio-ir,gpio_pin=17
#dtoverlay=gpio-ir-tx,gpio_pin=18

# Additional overlays and parameters are documented /boot/overlays/README

# Enable audio (loads snd_bcm2835)
dtparam=audio=on

# Automatically load overlays for detected cameras
start_x=1
```

```
# Automatically load overlays for detected DSI displays
display_auto_detect=1

# Enable DRM VC4 V3D driver
#dtoverlay=vc4-kms-v3d
max_framebuffers=2

# Disable compensation for displays with overscan
disable_overscan=1

[cm4]
# Enable host mode on the 2711 built-in XHCI USB controller.
# This line should be removed if the legacy DWC2 controller is required
# (e.g. for USB device mode) or if USB support is not required.
otg_mode=1

[all]

[pi4]
dtoverlay=vc4-fkms-v3d
# Run as fast as firmware / board allows
arm_boost=1

[all]
gpu_mem=256
```

C203 시연 시나리오

1. 홈페이지 접속 소행성
2. 회원가입 및 로그인
3. 마이페이지 학생 등록
4. 시뮬레이션 목록페이지
5. 시뮬레이션 상세페이지
6. 시뮬레이션 시작 버튼 클릭
7. 기기연결확인 페이지 OTP 생성
8. 기기에 OTP 입력
9. 연결확인 페이지 연결
10. 참여 학생 선택
11. 캐릭터선택
12. 카메라 연결상태 확인
13. 캐릭터 연동 확인
14. 앱 로그인
15. 시뮬레이션 시작
16. 시뮬레이션 미션 진행