포팅메뉴얼

1. 개요

1. 프로젝트 사용 도구

- 이슈 관리 : JIRA
- 형상 관리 : GITLAB
- 커뮤니케이션 : Notion, MatterMost
- 디자인 : Figma
- 영상포트폴리오 : Movavi, haiper (AI)
- CI/CD : Jenkins, Docker

2. 개발환경

- VS Code : 1.86
- IntelliJ : 242.21829.142
- Java : 17
- Node.js : 10.19.0
- SERVER : AWS EC2 Ubuntu 20.04.6 LTS
- DB : Mysql, Redis, MongoDB
- Android : expo 52
- AndroidStudio :

3. 외부 서비스

- Google Oauth2
- S3(Cloud Storage)

2. BackEnd 빌드

1. EC2 세팅

포트번호

컨테이너 이름	포트 번호 (외부 -> 내부)	서비스	특징
`config-server`	8888 → 8080	config server	
`eureka-server`	8761 → 8761	eureka server	
`api-gateway-blue`	8765 → 8765	gateway	blue-green무중단 배포
`api-gateway-green`	8766 → 8765	gateway	blue-green무중단 배포
`member-redis`	6379 → 6379	database	
`tournament-redis`	6380 → 6379	database	
`route-redis`	6381 → 6379	database	
`route-mongoDB`	27017 → 27017	database	
`mongo-express`	8081 → 8081	database	
`tournament-mysql`	3307 → 3306	database	
`member-mysql`	3306 → 3306	database	
`member-blue`	18010 → 8080	springboot	blue-green무중단 배포
`member-green`	18011 → 8080	springboot	blue-green무중단 배포

컨테이너 이름	포트 번호 (외부 -> 내부)	서비스	특징
`tournament-blue`	18020 → 8080	springboot	blue-green무중단 배포
`tournament-green`	18021 → 8080	springboot	blue-green무중단 배포
`route-blue`	18030 → 8080	springboot	blue-green무중단 배포
`route-green`	18031 → 8080	springboot	blue-green무중단 배포
`kafka-ui`	9000 → 8080	apache kafka	
`kafka-broker-1`	29092 -> 29092, 29094 -> 29094, 9092 (내부)	apache kafka	
`kafka-broker-2`	39092 -> 39092, 39094 -> 39094, 9092 (내부)	apache kafka	
`kafka-broker-3`	49092 -> 49092, 49094 -> 49094, 9092 (내부)	apache kafka	
`kafka-controller-1`	9092 (내부)	apache kafka	
`kafka-controller-2`	9092 (내부)	apache kafka	
`kafka-controller-3`	9092 (내부)	apache kafka	
`jenkins`	8080 -> 8080, 50000 (내부)	jenkins	
`nginx`	80 → 80, 443 → 443	nginx	

2. Config-Server

MSA를 위한 설정 파일

```
server:
  port: 18010
  servlet:
```

```
    session:
      cookie:
        name: JSESSIONID

spring:
  security:
    oauth2:
      client:
        registration:
          google:
            client-id: *****
            client-secret: *****
            redirect-uri: https://k11c207.p.ssafy.io/maon/membe
  application:
    name: member
  mvc:
    static-path-pattern: /static/**
  datasource:
    # 경로 수정
    url: jdbc:mysql://member-mysql:3306/member?serverTimezone=U
    driver-class-name: com.mysql.cj.jdbc.Driver
    username: *****
    password: *****
  jwt:
    secret: *****
    token:
      access-expiration-time: 86400000 # 24시간
      refresh-expiration-time: 604800000 # 7일
  hmac:
    key: *****
  jpa:
    show-sql: true
    hibernate:
      ddl-auto: update
    properties:
      hibernate:
```

```
        format_sql: true
  cloud:
    config:
      uri: http://config-server:8888
  redis:
    host: member-redis
    port: 6379
    password: *****
  config:
    import: s3.properties
  servlet:
    multipart:
      max-file-size: 20MB     # 파일 하나의 최대 크기 설정
      max-request-size: 20MB  # 요청 전체의 최대 크기 설정

eureka:
  client:
    serviceUrl:
      defaultZone: http://eureka-server:8761/eureka/

management:
  endpoints:
    web:
      exposure:
        include: health
        exclude: "*"

external-url:
  google-oauth: https://oauth2.googleapis.com/tokeninfo
  login-test: "https://k11c207.p.ssafy.io/maon/member/member/log
```

3. Docker 컨테이너

Database

```yaml
version: '3'
services:
  member-mysql:
    image: mysql:8.0
    container_name: member-mysql
    environment:
      - MYSQL_ROOT_PASSWORD=eastersumMoa2!
      - MYSQL_DATABASE=member
      - MYSQL_USER=eastersum
      - MYSQL_PASSWORD=eastersumMoa2!
    networks:
      - maonnet
    ports:
      - "3306:3306"
    volumes:
      - member-db-data:/var/lib/mysql
    restart: always

  member-redis:
    container_name: member-redis
    image: redis
    ports:
      - "6379:6379"
    networks:
      - maonnet
    volumes:
      - member-redis-data:/data
    restart: always

  tournament-mysql:
    image: mysql:8.0
    container_name: tournament-mysql
    environment:
      - MYSQL_ROOT_PASSWORD=eastersumMoa2!
```

```
      - MYSQL_DATABASE=tournament
      - MYSQL_USER=eastersum
      - MYSQL_PASSWORD=eastersumMoa2!
    networks:
      - maonnet
    ports:
      - "3307:3306"
    volumes:
      - tournament-db-data:/var/lib/mysql
    restart: always

  tournament-redis:
    container_name: tournament-redis
    image: redis
    ports:
      - "6380:6379"
    networks:
      - maonnet
    volumes:
      - tournament-redis-data:/data
    restart: always

  route-mongoDB:
    container_name: route-mongoDB
    image: mongo:latest
    ports:
      - "27017:27017"
    volumes:
      - route-mongoDB-data:/data/db
    networks:
      - maonnet
    restart: always

  route-redis:
    container_name: route-redis
    image: redis
```

```
    ports:
      - "6381:6379"
    networks:
      - maonnet
    volumes:
      - route-redis-data:/data
    restart: always

volumes:
  member-db-data:
  member-redis-data:
  tournament-db-data:
  tournament-redis-data:
  route-mongoDB-data:
  route-redis-data:

networks:
  maonnet:
    external: true
```

Config Server

```
version: '3.8'

services:
  app:
    build:
      context: .
      dockerfile: Dockerfile
    ports:
      - "8888:8080"  # 컨테이너 포트를 호스트 포트에 매핑
    environment:
```

```
      SPRING_PROFILES_ACTIVE: local  # 선택 사항: Spring 프로필 설정
    restart: always  # 재시작 정책
```

Service

docker-compose-blue 예시

```
version: "3"
services:
  # blue, green
  member-blue:
    build:
      context: ./back/member
      dockerfile: Dockerfile
    container_name: member-blue
    ports:
      - "18010:8080"
    networks:
      - maonnet
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - SPRING_CLOUD_CONFIG_URI=http://config-server:8888
      - SPRING_PROFILES_COLOR=blue

  tournament-blue:
    build:
      context: ./back/tournament
      dockerfile: Dockerfile
    container_name: tournament-blue
    ports:
      - "18020:8080"
    networks:
      - maonnet
    environment:
```

```
      - SPRING_PROFILES_ACTIVE=prod
      - SPRING_CLOUD_CONFIG_URI=http://config-server:8888
      - SPRING_PROFILES_COLOR=blue

  route-blue:
    build:
      context: ./back/route
      dockerfile: Dockerfile
    container_name: route-blue
    ports:
      - "18030:8080"
    networks:
      - maonnet
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - SPRING_CLOUD_CONFIG_URI=http://config-server:8888
      - SPRING_PROFILES_COLOR=blue

networks:
  maonnet:
    external: true
```

Dockerfile 예시

```
# Use an official OpenJDK runtime as a parent image
FROM openjdk:17-jdk-alpine

# Set the working directory in the container
WORKDIR /app

# Copy the JAR file to the container
COPY build/libs/*.jar /app/app.jar

# Expose the port
```

```
EXPOSE 18010

# Set the environment variable to activate the 'prod' profile
ENV SPRING_CLOUD_CONFIG_URI=http://config-server:8888
ENV SPRING_PROFILES_ACTIVE=prod

# Run the application
ENTRYPOINT ["java", "-jar", "app.jar"]
```

gateway

```
version: "3"
services:
  api-gateway-blue:
    build:
      context: ./back/gateway
      dockerfile: Dockerfile
    container_name: api-gateway-blue
    ports:
      - "8765:8765"
    networks:
      - maonnet
    restart: always
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - SPRING_CLOUD_CONFIG_URI=http://config-server:8888

networks:
  maonnet:
    external: true
```

eureka

```yaml
version: "3"
services:
# eureka, api-gateway는 blue-blue 전략을 사용하지 않는다.
  eureka-server:
    build:
      context: ./back/eureka
      dockerfile: Dockerfile
    container_name: eureka-server
    ports:
      - "8761:8761"
    networks:
      - maonnet
    restart: always
    environment:
      - SPRING_PROFILES_ACTIVE=prod
      - SPRING_CLOUD_CONFIG_URI=http://config-server:8888

networks:
  maonnet:
    external: true
```

Kafka

```yaml
version: "3.8"
services:
  kafka-ui:
    container_name: kafka-ui
    image: provectuslabs/kafka-ui:latest
    ports:
      - 9000:8080
    environment:
      KAFKA_CLUSTERS_0_NAME: local
      #KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: kafka-broker-1:29092,
```

```
      KAFKA_CLUSTERS_0_BOOTSTRAPSERVERS: k11c207.p.ssafy.io:2909
    depends_on:
      - kafka-controller-1
      - kafka-controller-2
      - kafka-controller-3
      - kafka-broker-1
      - kafka-broker-2
      - kafka-broker-3

  kafka-controller-1:
    image: apache/kafka:3.8.0
    container_name: kafka-controller-1
    environment:
      KAFKA_NODE_ID: 1
      KAFKA_PROCESS_ROLES: controller
      KAFKA_LISTENERS: CONTROLLER://:9093
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-controller-1:9093,
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"

  kafka-controller-2:
    image: apache/kafka:3.8.0
    container_name: kafka-controller-2
    environment:
      KAFKA_NODE_ID: 2
      KAFKA_PROCESS_ROLES: controller
      KAFKA_LISTENERS: CONTROLLER://:9093
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-controller-1:9093,
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"

  kafka-controller-3:
```

```
    image: apache/kafka:3.8.0
    container_name: kafka-controller-3
    environment:
      KAFKA_NODE_ID: 3
      KAFKA_PROCESS_ROLES: controller
      KAFKA_LISTENERS: CONTROLLER://:9093
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-controller-1:9093,
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"

  kafka-broker-1:
    image: apache/kafka:3.8.0
    container_name: kafka-broker-1
    ports:
      - 29092:29092
      - 29094:29094
    environment:
      KAFKA_NODE_ID: 4
      KAFKA_PROCESS_ROLES: broker
      KAFKA_LISTENERS: "PLAINTEXT://:29092,PLAINTEXT_HOST://:290
      KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka-broker-1:29
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: CONTROLLER:PLAINTEXT
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-controller-1:9093,
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 3
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 2
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
      KAFKA_MIN_INSYNC_REPLICAS: 2
    depends_on:
      - kafka-controller-1
```

```
      - kafka-controller-2
      - kafka-controller-3

  kafka-broker-2:
    image: apache/kafka:3.8.0
    container_name: kafka-broker-2
    ports:
      - 39092:39092
      - 39094:39094
    environment:
      KAFKA_NODE_ID: 5
      KAFKA_PROCESS_ROLES: broker
      KAFKA_LISTENERS: "PLAINTEXT://:39092,PLAINTEXT_HOST://:390
      KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka-broker-2:39
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: CONTROLLER:PLAINTEXT
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-controller-1:9093,
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 3
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 2
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
      KAFKA_MIN_INSYNC_REPLICAS: 2
    depends_on:
      - kafka-controller-1
      - kafka-controller-2
      - kafka-controller-3

  kafka-broker-3:
    image: apache/kafka:3.8.0
    container_name: kafka-broker-3
    ports:
      - 49092:49092
      - 49094:49094
```

```
    environment:
      KAFKA_NODE_ID: 6
      KAFKA_PROCESS_ROLES: broker
      KAFKA_LISTENERS: "PLAINTEXT://:49092,PLAINTEXT_HOST://:490
      KAFKA_ADVERTISED_LISTENERS: "PLAINTEXT://kafka-broker-3:49
      KAFKA_INTER_BROKER_LISTENER_NAME: PLAINTEXT
      KAFKA_CONTROLLER_LISTENER_NAMES: CONTROLLER
      KAFKA_LISTENER_SECURITY_PROTOCOL_MAP: CONTROLLER:PLAINTEXT
      KAFKA_CONTROLLER_QUORUM_VOTERS: 1@kafka-controller-1:9093,
      KAFKA_GROUP_INITIAL_REBALANCE_DELAY_MS: 0
      KAFKA_OFFSETS_TOPIC_REPLICATION_FACTOR: 3
      KAFKA_TRANSACTION_STATE_LOG_REPLICATION_FACTOR: 3
      KAFKA_TRANSACTION_STATE_LOG_MIN_ISR: 2
      KAFKA_LOG_DIRS: "/tmp/kraft-combined-logs"
      KAFKA_AUTO_CREATE_TOPICS_ENABLE: "false"
      KAFKA_MIN_INSYNC_REPLICAS: 2
    depends_on:
      - kafka-controller-1
      - kafka-controller-2
      - kafka-controller-3
```

Nginx

```
version: "3"
services:
  nginx:
    image: nginx
    container_name: nginx
    restart: always
    volumes:
      - /etc/nginx/blue-container.conf:/etc/nginx/blue-container
      - /etc/nginx/green-container.conf:/etc/nginx/green-contain
      - /etc/nginx/upstream.conf:/etc/nginx/upstream.conf
      - /etc/nginx/nginx.conf:/etc/nginx/nginx.conf
      - /etc/letsencrypt/live/k11c207.p.ssafy.io/fullchain.pem:/
```

```
      - /etc/letsencrypt/live/k11c207.p.ssafy.io/privkey.pem:/et
      - ./web:/usr/share/nginx/html
    ports:
      - 80:80
      - 443:443
    networks:
      - maonnet

networks:
  maonnet:
    external: true
```

nginx 설정

```
user  nginx;
worker_processes  auto;

# 에러 로그 레벨 설정
error_log  /var/log/nginx/error.log warn;
pid        /var/run/nginx.pid;

events {
    worker_connections  1024;
    multi_accept on;  # 다중 접속 허용
    use epoll;  # Linux에서 성능 향상
}

http {

    upstream blue {
      #API Gateway 인스턴스blue
      server api-gateway-blue:8765;
    }

    upstream green {
```

```
        #API Gateway 인스턴스 green
        server api-gateway-green:8765;
    }

    geo $upstream_env {
#       default "blue";
        include /etc/nginx/upstream.conf;
    }

    # 기본 설정
    include       mime.types;
    default_type  application/octet-stream;

    # 로깅 설정
    log_format  main  '$remote_addr - $remote_user [$time_local]
                      '$status $body_bytes_sent "$http_referer"
                      '"$http_user_agent" "$http_x_forwarded_for"
    access_log  /var/log/nginx/access.log  main;

    # 성능 최적화
    sendfile        on;
    tcp_nopush      on;
    tcp_nodelay     on;
    keepalive_timeout  65;
    types_hash_max_size 2048;

    # 버퍼 사이즈 설정
    client_body_buffer_size 10K;
    client_header_buffer_size 1k;
    client_max_body_size 8m;
    large_client_header_buffers 2 1k;

    # Gzip 압축
    gzip on;
    gzip_types text/plain text/css application/json application/
```

```
    # SSL 설정
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers on;
    ssl_ciphers ECDHE-ECDSA-AES128-GCM-SHA256:ECDHE-RSA-AES128-G
    ssl_session_cache shared:SSL:10m;
    ssl_session_timeout 10m;

    # Docker DNS resolver
    resolver 127.0.0.11 valid=5s ipv6=off;

    server {
      listen 80;
      server_name k11c207.p.ssafy.io;

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

    server {
        listen       443 ssl;
        server_name  k11c207.p.ssafy.io;

        # SSL 인증서
        ssl_certificate     /etc/ssl/fullchain.pem;
        ssl_certificate_key /etc/ssl/privkey.pem;

        # CORS 설정
        add_header 'Access-Control-Allow-Origin' '*';
        #add_header 'Access-Control-Allow-Methods' 'GET, POST, O
        #add_header 'Access-Control-Allow-Headers' 'DNT,User-Age

        # 보안 헤더
        add_header X-Frame-Options "SAMEORIGIN";
        add_header X-XSS-Protection "1; mode=block";
        add_header X-Content-Type-Options "nosniff";
```

```
        add_header Strict-Transport-Security "max-age=31536000;

        # 기본 웹 서비스
        location / {
            root   /usr/share/nginx/html;
            index  index.html index.htm;
            try_files $uri $uri/ /index.html;  # SPA 지원
            autoindex off;
        }

        # API Gateway 프록시
        location /maon/ {
            # 프록시 설정
            proxy_pass http://$upstream_env;

            # 프록시 헤더 설정
            proxy_set_header Host $host;
            proxy_set_header X-Real-IP $remote_addr;
            proxy_set_header X-Forwarded-For $proxy_add_x_forwar
            proxy_set_header X-Forwarded-Proto $scheme;


            # 타임아웃 설정
            proxy_connect_timeout 60s;
            proxy_send_timeout 300s;
            proxy_read_timeout 300s;

            # 버퍼 설정
            proxy_buffer_size 4k;
            proxy_buffers 4 32k;
            proxy_busy_buffers_size 64k;

            # 에러 처리
            proxy_next_upstream error timeout http_502 http_503

            # HTTP/1.1 지원
```

```
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
    }

    location /ws/ {
        # WebSocket 프록시 설정
        proxy_pass http://$upstream_env;  # $upstream_env는

        # WebSocket 관련 헤더 설정
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwar
        proxy_set_header X-Forwarded-Proto $scheme;

        # WebSocket 업그레이드 요청 처리
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';  # WebSocket

        # 타임아웃 설정
        proxy_connect_timeout 60s;
        proxy_send_timeout 300s;
        proxy_read_timeout 300s;

        # 버퍼 크기 설정
        proxy_buffer_size 4k;
        proxy_buffers 4 32k;
        proxy_busy_buffers_size 64k;

        # 에러 처리
        proxy_next_upstream error timeout http_502 http_503

        # HTTP/1.1 지원
        proxy_http_version 1.1;
    }
```

```
        # gateway 헬스체크
        location /actuator-blue/ {
          proxy_pass http://api-gateway-blue:8765/actuator/;

          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarde
          proxy_set_header X-Forwarded-Proto $scheme;

        }

        location /actuator-green/ {
          proxy_pass http://api-gateway-green:8765/actuator/;

          proxy_set_header Host $host;
          proxy_set_header X-Real-IP $remote_addr;
          proxy_set_header X-Forwarded-For $proxy_add_x_forwarde
          proxy_set_header X-Forwarded-Proto $scheme;
        }

        # 에러 페이지
        error_page 500 502 503 504 /50x.html;
        location = /50x.html {
            root /usr/share/nginx/html;
        }

        # 악성 요청 차단
        location = /favicon.ico {
            access_log off;
            log_not_found off;
        }

        # 숨김 파일 접근 차단
        location ~ /\. {
            deny all;
```

```
            access_log off;
            log_not_found off;
        }
    }

    # Rate limiting 설정
    limit_req_zone $binary_remote_addr zone=api_limit:10m rate=
    limit_conn_zone $binary_remote_addr zone=addr:10m;

    # badbot 라이브러리로 악성 봇 차단
    include nginx-badbot-blocker/blacklist.conf;
    include nginx-badbot-blocker/blockips.conf;
}
```

Jenkins

```
version: "3"
services:
  jenkins:
    image: jenkins/jenkins:lts
    user: root
    privileged: true
    container_name: jenkins
    volumes:
      - /etc/nginx/green-container.conf:/etc/nginx/green-contai
      - /etc/nginx/blue-container.conf:/etc/nginx/blue-containe
      - /etc/nginx/upstream.conf:/etc/nginx/upstream.conf
      - ./jenkins:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
      - /usr/bin/docker:/usr/bin/docker
    ports:
      - 8080:8080
    networks:
      - maonnet
```

```
networks:
  maonnet:
    external: true
```

Jenkins-Pipeline

```
pipeline {
    agent any

    environment {
        GITLAB_CREDENTIALSID = 'gitlab_login'
    }

    stages {
        stage('git checkout') {
            steps {
                git credentialsId: GITLAB_CREDENTIALSID, branch
            }
        }

        stage('Set Deployment Environment') {
            steps {
                script {
                    def runningContainers = getRunningContainers

                    CURRENT_ENV = "blue"
                    NEXT_ENV = "green"

                    // 색상에 따라 현재 및 다음 환경 결정
                    def greenContainers = runningContainers.find
                    def blueContainers = runningContainers.findA

                    // 실행 중인 컨테이너에 따라 환경 설정
                    if (greenContainers.size() > blueContainers
```

```
                        CURRENT_ENV = "green"
                        NEXT_ENV = "blue"
                    } else if (blueContainers.size() > 0 && gree
                        CURRENT_ENV = "blue"
                        NEXT_ENV = "green"
                    } else {
                        echo "Both environments are running or n
                    }

                    // 최종 결정된 현재 환경과 다음 환경 출력
                    echo "Current Environment: ${CURRENT_ENV}"
                    echo "Next Environment: ${NEXT_ENV}"

                    // 환경 변수 설정
                    env.SPRING_PROFILES_COLOR = NEXT_ENV
                }
            }
        }

        stage('Build and Deploy to Next Environment') {
            steps {
                script {
                    try {
                        def runningContainers = getRunningContai

                        // Eureka 실행 확인 및 시작
                        if (!runningContainers.any { it.contains
                            echo "Eureka is not running. Startin
                            startEureka()
                        } else {
                            echo "Eureka is already running."
                        }

                        // 게이트웨이 변경 확인 및 빌드
                        // def changes = sh(script: 'git diff --
                        // if (changes.contains('./back/gateway
```

```
                        //     echo 'Changes detected in ./back/
                        // }
                        echo "================ gateway build and
                        deployGateway()

                        // 다음 환경에 배포
                        echo "================ service build and
                        deployServices()

                        // 헬스 체크 및 NGINX 설정 변경
                        echo "================ health check star
                        // performHealthCheck()

                        echo "================ update nginx conf
                        updateNginxConfiguration()

                        // 현재 환경 종료
                        shutdownCurrentEnvironment()

                    } catch (Exception e) {
                        echo "Deployment failed: ${e.getMessage(
                        currentBuild.result = 'FAILURE'
                        error("Deployment failed")
                    }
                }
            }
        }

        stage('Cleanup Unused Docker Images') {
            steps {
                script {
                    sh 'docker image prune -f'
                    echo "Unused Docker images cleaned up."
                }
            }
        }
```

```
    }

    post {
        failure {
            echo "Pipeline failed. Check the logs for details."

            script {
                def Author_ID = sh(script: "git show -s --pretty
                def Author_Name = sh(script: "git show -s --pret
                mattermostSend (color: 'danger',
                message: "빌드 실패: ${env.JOB_NAME} #${env.BUILD
                endpoint: 'https://meeting.ssafy.com/hooks/ij1z
                channel: 'jenkins-build'
                )
            }
        }
        success {
            echo "Pipeline completed successfully. New environme

            script {
                def Author_ID = sh(script: "git show -s --pretty
                def Author_Name = sh(script: "git show -s --pret
                mattermostSend (color: 'good',
                message: "빌드 성공: ${env.JOB_NAME} #${env.BUILD
                endpoint: 'https://meeting.ssafy.com/hooks/ij1z
                channel: 'jenkins-build'
                )
            }
        }
    }
}

// 함수 정의
def getRunningContainers() {
    return sh(script: "docker ps --format '{{.Names}}'", returnS
}
```

```
def startEureka() {
    dir("./back/eureka") {
        if (fileExists('.')) {
            sh 'chmod +x gradlew'
            def buildResult = sh(script: './gradlew clean build
            if (buildResult != 0) {
                error("Build failed for eureka service.")
            }
        } else {
            error("Directory not found for eureka service.")
        }
    }
    def upEurekaResult = sh(script: "docker-compose -f \"docker
    if (upEurekaResult != 0) {
        error("Failed to deploy Eureka.")
    }
    echo "Eureka is now running."
}

def deployGateway() {
    // Gateway 빌드
    dir("./back/gateway") {
        if (fileExists('.')) {
            sh 'chmod +x gradlew'
            def buildResult = sh(script: './gradlew clean build
            if (buildResult != 0) {
                error("Build failed for gateway service.")
            }
        } else {
            error("Directory not found for gateway service.")
        }
    }
    echo "gateway build ok"

    // 기존 next 환경의 gateway down
```

```
    sh "docker-compose -f docker-compose-gateway-${NEXT_ENV}.yml
    echo "gateway ${NEXT_ENV} down ok"

    // next 환경에만 gateway 배포
    def upResult = sh(script: "docker-compose -f docker-compose-

    if (upResult != 0) {
        error("Failed to deploy gateway to ${NEXT_ENV} environme
    }
    echo "Successfully deployed gateway to ${NEXT_ENV} environme
}

def deployServices() {
    // 각 서비스 빌드
    ["member", "route", "tournament"].each { service ->
        echo "Building ${service} service"
        dir("./back/${service}") {
            if (fileExists('.')) {
                sh 'chmod +x gradlew'
                def buildResult = sh(script: './gradlew clean bu
                if (buildResult != 0) {
                    error("Build failed for ${service} service."
                }
            } else {
                error("Directory not found for ${service} servic
            }
        }
    }

    // 다음 환경에 배포
    def downResult = sh(script: "docker-compose -f \"docker-comp
    if (downResult != 0) {
        error("Failed to stop containers in the ${NEXT_ENV} envi
    }
    echo "Successfully stopped containers in ${NEXT_ENV}"
```

```
    def upResult = sh(script: "SPRING_PROFILES_COLOR=${env.SPRI
    if (upResult != 0) {
        error("Failed to deploy to ${NEXT_ENV} environment")
    }
    echo "Successfully deployed to ${NEXT_ENV}"
}

def performHealthCheck() {
    def healthCheckAttempts = 10
    def healthCheckPassed = false
    for (int i = 1; i <= healthCheckAttempts; i++) {
        echo "Performing health check attempt ${i}/${healthChec
        def response = sh(script: "curl -s -o /dev/null -w '%{ht

        echo "Health Check Response: ${response}"

        if (response == '200') {
            echo "Health check passed with status code: ${respo
            healthCheckPassed = true
            break
        } else {
            echo "Health check failed with status code: ${respo
            sleep 10
        }
    }

    if (!healthCheckPassed) {
        error "Health check failed after ${healthCheckAttempts}
    }
}

def updateNginxConfiguration() {
    def nginxConfigFile = "/etc/nginx/${NEXT_ENV}-container.con
    def nginxUpstreamFile = "/etc/nginx/upstream.conf"

    def cpResult = sh(script: "cp ${nginxConfigFile} ${nginxUps
```

```
    if (cpResult != 0) {
        error("Failed to update NGINX configuration.")
    }
    echo "Successfully updated NGINX configuration from ${nginx

    // nginx 컨테이너 재시작
    def restartResult = sh(script: "docker restart nginx", retu
    if (restartResult != 0) {
        error("Failed to restart NGINX container.")
    }
    echo "Successfully restarted NGINX container"
}

def shutdownCurrentEnvironment() {
    echo "Shutting down ${CURRENT_ENV} environment"
    def downCurrentResult = sh(script: "docker-compose -f \"doc
    if (downCurrentResult != 0) {
        error("Failed to shut down ${CURRENT_ENV} environment")
    }
    echo "${CURRENT_ENV} environment is down, ${NEXT_ENV} is no
}
```

3. Android Build

1. 빌드

1. WSL Ubuntu 가상환경 생성

(expo 로컬 빌드가 Window에서는 지원하지 않음)

```
wsl --install
```

2. Git Clone 으로 React-Native 프로젝트 받기

```
git clone https://lab.ssafy.com/s11-final/S11P31C207.gi
```

3. expo prebuild로 세부 설정 진행

```
npx expo prebuild clean
npx expo prebuild
```

생성된 /android/app/src/main/AndroidManifest.xml 파일에

Google 맵 키, 딥링크를 위한 sheme 생성

4. expo local 빌드 진행

```
eas build --platform android --profile production --local
```

eas.json (빌드 설정 파일)

```
{
  "cli": {
    "version": ">= 12.6.1",
    "appVersionSource": "remote"
  },
  "build": {
    "development": {
      "developmentClient": true,
      "distribution": "internal",
      "android": {
        "buildType": "apk"
      },
      "env": {
        "NODE_ENV": "development",
        "EXPO_DEBUG": "true"
```

```
      }
    },
    "preview": {
      "distribution": "internal"
    },
    "production": {
      "autoIncrement": true,
      "env": {
        "NODE_ENV": "production"
      },
      "android": {
        "buildType": "apk"
      }
    }
  },
  "submit": {
    "production": {}
  }
}
```

2. 배포

1. 생성된 apk 를 aws S3에 배포

S3 Url을 가지고 웹의 QR코드 주소를 업데이트