Pawly 포팅메뉴얼

🐾 개발환경

FrontEnd

- **Node.js** `20.15.0`
- **TypeScript** `5.2.2`
- **vite** `5.2.10`
  - **vite-plugin-pwa** `0.20.5`
- **React** `18.3.1`
  - **zustand** `5.0.0`
  - **react-query** `5.59.15`
- **emotion css** `11.13.3`
- **axios** `1.7.7`
- **firebase** `11.0.1`

BackEnd

- **Java**
  - **Java OpenJDK** `17.0.12`
  - **Spring Boot** `3.3.4`
    - **Spring Data JPA** `3.3.4`
    - **Spring Data redis** `3.3.4`
    - **Spring Security** `6.6.3`
    - **OAuth2.0** `6.3.3`
    - **Lombok** `1.18.34`
- **Python**
  - **Python** `3.9`
  - **Flask** `3.0.3`
  - **requests** `2.32.3`
  - **openai** `1.53.0`
  - **pillow** `11.0.0`

- JWT `0.12.3`
- AWS S3 Bucket Cloud `2.2.6`
- firebase `7.3.0`
- Gradle `8.10`

UI/UX

- Figma

IDE

- IntelliJ `2024-01`
- Visual Studio Code `1.94.1`

Server 배포 환경

- AWS EC2 `ubuntu 20.04.6 LTS`
- Docker `27.2.0`
- Docker Compose `2.29.2`
- Nginx `1.18.0`
- SSL
- Docker Hub

CI/CD

- Jenkins `2.475`

DB

- MySQL `8.0.38`
- redis `7.4.0`
- AWS S3

Collaboration

형상관리

- GitLab

커뮤니케이션

- Mattermost
- Notion

이슈관리

- Jira

🐾 환경 변수 설정

[FrontEnd]

🖨️ .env

```
VITE_BACKEND_URL=${VITE_BACKEND_URL}
VITE_FB_API_KEY=${VITE_FB_API_KEY}
VITE_FB_AUTH_DOMAIN=${VITE_FB_AUTH_DOMAIN}
VITE_FB_PROJECT_ID=${VITE_FB_PROJECT_ID}
VITE_FB_STORAGE_BUCKET=${VITE_FB_STORAGE_BUCKET}
VITE_FB_MESSAGING_SENDER_ID=${VITE_FB_MESSAGING_SENDER_ID}
VITE_FB_APP_ID=${VITE_FB_APP_ID}
VITE_FB_MEASUREMENT_ID=${VITE_FB_MEASUREMENT_ID}
VITE_VAPID_KEY=${VITE_VAPID_KEY}
```

[BackEnd]

🖨️ application-prod.yml

```
spring:
  application:
    name: pawly
  servlet:
    multipart:
      max-request-size: 50MB
      max-file-size: 50MB
```

```
  datasource:
    driver-class-name: com.mysql.cj.jdbc.Driver
    url: jdbc:mysql://${MYSQL_HOST}:${MYSQL_PORT}/${MYSQL_DBNAME}?useSSL=false&serverTimezone=As
    username: ${MYSQL_USERNAME}
    password: ${MYSQL_PASSWORD}
  jpa:
    properties:
      hibernate.format_sql: true
      dialect: org.hibernate.dialect.MySQL8InnoDBDialect
      naming:
        physical-strategy: org.hibernate.boot.model.naming.PhysicalNamingStrategyStandardImpl
  data:
    redis:
      host: ${REDIS_HOST}
      port: ${REDIS_PORT}
      password: ${REDIS_PASSWORD}
      username: ${REDIS_USERNAME}
  security:
    oauth2:
      client:
        registration:
          kakao:
            client-id: ${KAKAO_CLIENT_ID}
            client-secret: ${KAKAO_CLIENT_SECRET}
            redirect-uri: ${KAKAO_REDIRECT_URI}
            authorization-grant-type: authorization_code
            client-authentication-method: client_secret_post
            client-name: Kakao
            scope:
              - profile_nickname
              - account_email
          google:
            clientId: ${GOOGLE_CLIENT_ID}
            client-secret: ${GOOGLE_CLIENT_SECRET}
            redirect-uri: ${GOOGLE_REDIRECT_URI}
            client-name: Google
            scope: profile, email
        provider:
          kakao:
            authorization-uri: https://kauth.kakao.com/oauth/authorize
            token-uri: https://kauth.kakao.com/oauth/token
            user-info-uri: https://kapi.kakao.com/v2/user/me
            user-name-attribute: id

  mvc:
    async:
      request-timeout: 300000

logging:
  level:
    org.springframework.security: DEBUG
    org.springframework.web: DEBUG
    org.springframework.security.oauth2: DEBUG
    org.springframework.web.servlet: DEBUG
    org.springframework.web.client.RestTemplate: DEBUG

frontend:
  url: ${FRONT_DOMAIN}
```

```
oauth2:
  baseUrl: ${OAUTH2_DOMAIN}

cors:
  allowed-origin: http://localhost:8080, http://localhost:3000, http://127.0.0.1:3000, http://lo
  allowed-methods: '*'

jwt:
  access-secret: ${JWT_SECRET}
  access-expiration: ${JWT_ACCESS_TOKEN_EXPIRATION:604800000} # 7일
  refresh-expiration: ${JWT_REFRESH_TOKEN_EXPIRATION:1209600000} # 1,209,600,000 ms = 14일
  oauth-expiration: ${JWT_REFRESH_TOKEN_EXPIRATION:600000} # 10분

cloud:
  aws:
    s3:
      bucketName: ${S3_BUCKET_NAME}
      path:
        asset: asset/
        letter: letter/
    credentials:
      accessKey: ${S3_ACCESS_KEY}
      secretKey: ${S3_SECRET_KEY}
    region.static: ap-northeast-2
    stack.auto: false

flask:
  url: ${FLASK_DOMAIN}
```

🖨️ .env (Flask)

```
OPEN_API_KEY=${OPEN_API_KEY}
DEEPL_API_KEY=${DEEPL_API_KEY}
```

🐾 배포 환경 설정

0. 초기 세팅

1. EC2 접속

```
# sudo ssh -i [pem키 위치] [접속 계정]@[접속할 도메인]
$ sudo ssh -i K11D104T.pem ubuntu@k11d104.p.ssafy.io
```

2. Docker & Docker Engine 설치
3. Docker Compose 설치

1. Docker 컨테이너 생성

: 백엔드 Spring 서버, Dall-e 이미지 생성 및 배경제거 Flask 서버, 프론트 엔드 React, mysql, redis, jenkins

- `sudo docker ps` 결과

```
ubuntu@ip-172-26-9-71:~$ sudo docker ps
CONTAINER ID   IMAGE              COMMAND                  CREATED        STATUS        PORTS                                                    NAMES
21264311fc8a   ca700a74dbe7       "java -jar -Dspring.…"   19 hours ago   Up 19 hours   0.0.0.0:8081->8080/tcp, [::]:8081->8080/tcp              pawly-8081
3506ae854696   ppmm98/flask:latest "python /app/app.py"    24 hours ago   Up 24 hours   0.0.0.0:5000->5000/tcp, :::5000->5000/tcp                flask-app-container
2fdf252860b7   19c8ebb5570e       "docker-entrypoint.s…"   6 days ago     Up 6 days     0.0.0.0:5173->5173/tcp, :::5173->5173/tcp                pawly-frontend
2ef3d4b8348f   jenkins/jenkins    "/usr/bin/tini -- /u…"   6 days ago     Up 3 days     0.0.0.0:8080->8080/tcp, :::8080->8080/tcp, 50000/tcp     jenkins-container
123e2861096f   mysql:8.0.38       "docker-entrypoint.s…"   7 days ago     Up 7 days     0.0.0.0:3306->3306/tcp, :::3306->3306/tcp, 33060/tcp     mysql-container
fe75baefc586   redis              "docker-entrypoint.s…"   7 days ago     Up 7 days     0.0.0.0:6379->6379/tcp, :::6379->6379/tcp                redis-container
```

`/home/ubuntu/Dockerfiles` 경로에 docker-compose 파일 모아둠

```
$ tree ./Dockerfiles
./Dockerfiles
├── 📁jenkins
│  └── docker-compose.yml
├── 📁mysql
│  └── docker-compose.yml
└── 📁redis
  └── docker-compose.yml
```

MySQL

볼륨 생성 `$ docker volume create mysql-volumne`

```
services:
  mysql:
    image: mysql:8.0.38
    container_name: mysql-container
    restart : always
    ports:
      - "3306:3306"
    volumes:
      - /mysql-volume:/var/lib/mysql
    environment:
      MYSQL_DATABASE: ${MYSQL_DBNAME}
      MYSQL_ROOT_PASSWORD: ${MYSQL_ROOT_PASSWORD}
      TZ: "Asia/Seoul"
```

Redis

볼륨 생성 `$ docker volume create redis-volume`

```
services:
  redis:
    image: redis
    container_name: redis-container
    ports:
      - "6379:6379"
    command: redis-server --requirepass ${REDIS_PASSWORD}
    volumes:
      - /redis-volume:/data
    restart: on-failure
```

Jenkins

`$ cd /home/ubuntu && mkdir jenkins-backup`

`$ sudo chown 1000 /home/ubuntu/jenkins-backup`

```
services:
   jenkins:
     image: jenkins/jenkins
     container_name: jenkins-container
     ports:
       - "8080:8080"
     volumes:
       - /home/ubuntu/jenkins-backup:/var/jenkins_home
       - /var/run/docker.sock:/var/run/docker.sock
```

```
      - /usr/bin/docker:/usr/bin/docker
    environment:
      TZ: "Asia/Seoul"
```

- 젠킨스 컨테이너 비밀번호 확인

  `sudo docker exec jenkins-container cat /var/jenkins_home/secrets/initailAdminPassword`

- 실행

  `$ sudo docker compose up -d`

- 컨테이너 접속

  `$ sudo docker exec -it [컨테이너 이름] bash`

2. Nginx 설치 + SSL 인증키 발급

1. Nginx 설치

   `$ sudo apt update && sudo apt upgrade`

   `$ sudo apt install nginx`

   `$ sudo service nginx start`

2. Encrypt, Certbot 설치

   `$ sudo apt-get install letsencrypt`

   `$ sudo apt-get install certbot python3-certbot-nginx`

3. SSL 인증서 발급

```
# Certbot 동작 (nginx 중지하고 해야함)
$ sudo systemctl stop nginx

# Nginx 상태확인 & 80번 포트 확인
$ sudo service nginx status
$ netstat -na | grep '80.*LISTEN'

# SSL 인증서 발급 (인증서 적용 및 .pem 키 발급)
$ sudo certbot --nginx
$ sudo letsencrypt certonly --standalone -d k11d104.p.ssafy.io

# 설치한 인증서 확인 및 위치 확인
$ sudo certbot certificates

# nginx 설정 적용
# nginx 재시작
$ sudo service nginx restart
$ sudo systemctl reload nginx
```

🖨️ Nginx conf 설정

- service-url.inc 파일 생성

  `$ sudo vim /etc/nginx/conf.d/service-url.inc`

```
set $service_url http://127.0.0.1:8081;
```

- nginx 설정파일

  `$ sudo vim /etc/nginx/sites-available/default`

  + https (SSL 키 적용), `service-url.inc`를 통한 무중단 배포 진행

```
server {

    listen 443 ssl;
```

```
    listen [::]:443 ssl;
    server_name pawly.o-r.kr;

    ssl_certificate /etc/letsencrypt/live/pawly.o-r.kr/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/pawly.o-r.kr/privkey.pem;
    include /etc/letsencrypt/options-ssl-nginx.conf;
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem;

    root /var/www/html;
    index index.html index.htm index.nginx-debian.html;
    include /etc/nginx/conf.d/service-url.inc;

    # frontend
    location / {
        proxy_pass http://pawly.o-r.kr:5173;
        proxy_set_header Host $host:$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;
        proxy_cookie_path / "/";
    }

    # backend - spring
    location /api {
        proxy_pass $service_url;
        proxy_set_header Host $host;
        proxy_set_header X-Forwarded-Host $server_name;
        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;
        proxy_set_header Cookie $http_cookie;
        proxy_pass_header Set-Cookie;
        proxy_cookie_path / "/; HttpOnly; Secure; SameSite=None";
        proxy_redirect off;
    }

    # oauth2
    location ~ ^/(oauth2|login/oauth2) {
        proxy_pass $service_url;
        proxy_set_header Host $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;
    }

    # backend - flask
    location /flask {
        proxy_pass http://localhost:5000;
        proxy_set_header Host $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;
        proxy_set_header Cookie $http_cookie;
        proxy_pass_header Set-Cookie;
        proxy_cookie_path / "/; HttpOnly; Secure; SameSite=None";
        proxy_redirect off;
    }
}
```

```
server {

        root /var/www/html;

        server_name k11d104.p.ssafy.io;

        index index.html index.htm index.nginx-debian.html;
        include /etc/nginx/conf.d/service-url.inc;

        # frontend
        location / {
                # First attempt to serve request as file, then
                # as directory, then fall back to displaying a 404.
                proxy_pass http://pawly.o-r.kr:5173;
                proxy_set_header Host $host:$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;
                proxy_cookie_path / "/";
        }

        # backend - spring
        location /api {
                proxy_pass $service_url;

                proxy_set_header Host $host;
                proxy_set_header X-Forwarded-Host $server_name;
                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;

                proxy_set_header Cookie $http_cookie;
                proxy_pass_header Set-Cookie;
                proxy_cookie_path / "/; HttpOnly; Secure; SameSite=None";

                proxy_redirect off;
        }

        location ~ ^/(oauth2|login/oauth2) {
                proxy_pass $service_url;
                proxy_set_header Host $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;

        }
        # backend - flask
        location /flask {
                proxy_pass http://localhost:5000;  # HTTPS로 Flask 서버에 접근
                proxy_set_header Host $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;

                proxy_set_header Cookie $http_cookie;
                proxy_pass_header Set-Cookie;
                proxy_cookie_path / "/; HttpOnly; Secure; SameSite=None";
```

```
                proxy_redirect off;
        }
    listen [::]:443 ssl ipv6only=on; # managed by Certbot
    listen 443 ssl; # managed by Certbot

    # k11d104.p.ssafy.io 인증서 설정
    ssl_certificate /etc/letsencrypt/live/k11d104.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k11d104.p.ssafy.io/privkey.pem;

    include /etc/letsencrypt/options-ssl-nginx.conf; # managed by Certbot
    ssl_dhparam /etc/letsencrypt/ssl-dhparams.pem; # managed by Certbot
}
server {

        listen 80 default_server;
        listen [::]:80 default_server;

        server_name pawly.o-r.kr;
    return 301 https://$host$request_uri; # managed by Certbot
}
```

- S3에 파일 업로드 시 용량 제한 늘리기

  `$ sudo vim /etc/nginx/nginx.conf`

  해결하기 위해 http block에 아래의 옵션 추가

  ```
  client_max_body_size 50M;
  ```

- nginx 재시작 : 파일 수정 사항 적용

  `$ sudo systemctl restart nginx`

- nginx 로그 확인

  `$ cd /var/log/nginx`

  └ access.log, error.log 존재

3. Jenkins 설정

🖨️ 젠킨스 파이프라인 스크립트

: 특정 브랜치(backend, frontend)를 추적하여 자동 배포가 진행되도록 한다.

post{} 는 mattermost 알림을 위한 설정

▼ 백엔드

최초 1회 `firebase-service-account.json` 파일 복사를 위해 해당 파이프라인으로 진행

```
stage('application.yml and Firebase Config Download') {
            steps {
                withCredentials([
                    file(credentialsId: 'application-prod.yml', variable: 'applicationymlFile
                    file(credentialsId: 'firebase-service-account.json', variable: 'firebase(
                ]) {
                    script {
                        // application-prod.yml 파일 복사
                        sh '[ -f /var/jenkins_home/workspace/pawly-backend/backend/src/main/r
                        sh 'cp $applicationymlFile /var/jenkins_home/workspace/pawly-backend/

                        // firebase-service-account.json 파일 복사
                        sh '[ -f /var/jenkins_home/workspace/pawly-backend/backend/src/main/r
                        sh 'cp $firebaseConfigFile /var/jenkins_home/workspace/pawly-backend/
```

```
                    }
                }
            }
        }
```

이후 변경된 파이프 라인

```
pipeline {
    agent any
    stages {
        stage('Git Clone') {
            steps {
                git branch: 'backend',
                credentialsId: 'gitlab',
                url: 'https://lab.ssafy.com/s11-final/S11P31D104.git'
            }
            post {
                failure {
                    echo 'Repository clone failure !'
                }
                success {
                    echo 'Repository clone success !'
                }
            }
        }
        stage('application.yml and Firebase Config Download') {
            steps {
                withCredentials([
                    file(credentialsId: 'application-prod.yml', variable: 'applicationymlFile
                    file(credentialsId: 'firebase-service-account.json', variable: 'firebaseC
                ]) {
                    script {
                        // application-prod.yml 파일 복사
                        sh 'rm /var/jenkins_home/workspace/pawly-backend/backend/src/main/res
                        sh 'cp $applicationymlFile /var/jenkins_home/workspace/pawly-backend/

                        // firebase-service-account.json 파일 복사
                        sh 'rm /var/jenkins_home/workspace/pawly-backend/backend/src/main/res
                        sh 'cp $firebaseConfigFile /var/jenkins_home/workspace/pawly-backend/
                    }
                }
            }
        }
        stage('BE-Build') {
            steps {
                dir('/var/jenkins_home/workspace/pawly-backend/backend/') {
                    sh 'pwd'
                    sh 'ls -al'
                    sh 'chmod +x ./gradlew'
                    sh 'chmod +x ./gradlew.bat'
                    sh 'java --version'
                    sh './gradlew clean build -x test'
                }
            }
        }
        stage('Docker Hub Login') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'DOCKERHUB_USER', passwordVa
```

```
                    sh 'echo "$DOCKER_PASSWORD" | docker login -u $DOCKER_USERNAME --passwor
                }
            }
        }
        stage('Docker Build and Push') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'DOCKER_REPO', passwordVari
                    sh 'cd ./backend && docker build -f Dockerfile -t $DOCKER_USER/$DOCKER_PR
                    sh 'cd ./backend && docker push $DOCKER_USER/$DOCKER_PROJECT'
                    echo 'docker push Success!!'
                }
            }
        }
        stage('BE Deploy to EC2') {
            steps {
                sshagent(credentials: ['ssh-key']) {
                    withCredentials([string(credentialsId: 'EC2_SERVER_IP', variable: 'IP')])
                        sh 'ssh -o StrictHostKeyChecking=no ubuntu@$IP "sudo sh deploy.sh"'
                    }
                }
            }
        }
    }
    post {
        always {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).tr
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true)
                def Commit_Message = sh(script: "git log -1 --pretty=%B", returnStdout: true)
                def Build_Status = currentBuild.result ?: 'SUCCESS'
                def Status_Color = Build_Status == 'SUCCESS' ? 'good' : (Build_Status == 'UNS
                def Status_Text = Build_Status == 'SUCCESS' ? '빌드 성공' : (Build_Status == '

                def branchName = sh(script: "git rev-parse --abbrev-ref HEAD", returnStdout:

                def previousCommit = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT ?: 'HEAD~1'
                def allCommits = sh(script: "git log --pretty=format:'%h - %s (%an)' $previou
                def formattedCommits = allCommits.split('\\n').collect { line ->
                    def escapedLine = line.replaceAll("([\\[\\]\\(\\)])", '\\\$1')
                    "• ${escapedLine}"
                }.join('\\n')

                def message = """
                    #### 💻BE $Status_Text
                    **빌드 번호** $env.JOB_NAME #$env.BUILD_NUMBER
                    **브랜치:** $branchName
                    **작성자:** $Author_ID ($Author_Name)
                    **빌드 URL:** [Details]($env.BUILD_URL)
                    **포함된 커밋:**
                    $formattedCommits
                """.stripIndent()
                mattermostSend(
                    color: Status_Color,
                    message: message,
                    endpoint: 'https://meeting.ssafy.com/hooks/ussrtiwmpty7fro89skiommr9a',
                    channel: 'd104-cicd'
                )
            }
```

```
        }
    }
}
```

▼ **프론트엔드**

```
pipeline {
    agent any
    environment {
        PATH = "/usr/local/bin:/usr/bin:$PATH"
    }
    stages {
        stage('Git Clone') {
            steps {
                git branch: 'frontend',
                credentialsId: 'gitlab',
                url: 'https://lab.ssafy.com/s11-final/S11P31D104.git'
            }
        }
        stage('.env download') {
            steps {
                withCredentials([file(credentialsId: 'REACT_ENV', variable: 'ENV_FILE')]) {
                    script {
                        sh 'cp $ENV_FILE /var/jenkins_home/workspace/pawly-frontend/frontend/
                    }
                }
            }
        }
        stage('FE-Build') {
            steps {
                dir('/var/jenkins_home/workspace/pawly-frontend/frontend/') {
                    sh 'npm install'
                    sh 'npm run build'
                }
            }
        }
        stage('Docker Hub Login'){
            steps{
                withCredentials([usernamePassword(credentialsId: 'DOCKER_FE_USER', passwordVa
                    sh 'echo "$DOCKER_PASSWORD" | docker login -u $DOCKER_USERNAME --passwor
                }
            }
        }
        stage('Docker Build and Push') {
            steps {
                withCredentials([usernamePassword(credentialsId: 'DOCKER_FE_REPO', passwordVa
                    sh 'set -o allexport; . $ENV_FILE; set +o allexport'
                    sh 'cd ./frontend && docker build -f Dockerfile -t $DOCKER_USER/$DOCKER_P
                    sh 'cd ./frontend && docker push $DOCKER_USER/$DOCKER_PROJECT'
                }
                echo 'docker push Success!!'
            }
        }
        stage('FE Deploy to EC2') {
            steps {
                sshagent(credentials: ['ssh-key']) {
                  withCredentials([string(credentialsId: 'EC2_SERVER_IP', variable: 'IP')]) {
                      sh 'ssh -o StrictHostKeyChecking=no ubuntu@$IP "sudo sh deploy-frontend
```

```
                    }
                }
            }
        }
    }

    post {
        always {
            script {
                def Author_ID = sh(script: "git show -s --pretty=%an", returnStdout: true).tr
                def Author_Name = sh(script: "git show -s --pretty=%ae", returnStdout: true)
                def Commit_Message = sh(script: "git log -1 --pretty=%B", returnStdout: true)
                def Build_Status = currentBuild.result ?: 'SUCCESS'
                def Status_Color = Build_Status == 'SUCCESS' ? 'good' : (Build_Status == 'UNS
                def Status_Text = Build_Status == 'SUCCESS' ? '빌드 성공' : (Build_Status == '

                def branchName = sh(script: "git rev-parse --abbrev-ref HEAD", returnStdout:

                //def allCommits = sh(script: "git log --pretty=format:'%h - %s (%an)' $env.(
                def previousCommit = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT ?: 'HEAD~1' // 이전 커
                def allCommits = sh(script: "git log --pretty=format:'%h - %s (%an)' $previou
                def formattedCommits = allCommits.split('\\n').collect { line ->
                    def escapedLine = line.replaceAll("([\\[\\]\\(\\)])", '\\\$1')
                    "• ${escapedLine}"
                }.join('\\n')

                def message = """
                    #### 🎨FE $Status_Text
                    **빌드 번호** $env.JOB_NAME #$env.BUILD_NUMBER
                    **브랜치:** $branchName
                    **작성자:** $Author_ID ($Author_Name)
                    **빌드 URL:** [Details]($env.BUILD_URL)
                    **포함된 커밋:**
                    $formattedCommits
                """.stripIndent()
                mattermostSend(
                    color: Status_Color,
                    message: message,
                    endpoint: 'https://meeting.ssafy.com/hooks/ussrtiwmpty7fro89skiommr9a',
                    channel: 'd104-cicd'
                )
            }
        }
    }
}
```

🔑 Credential 관리

빌드에 필요한 env 파일들을 저장해두고 배포 시 파일을 옮겨 서버에 올린다.

Credentials

T	P	Store ↓	Domain	ID	Name
		System	(global)	gitlab_token	GitLab API token
		System	(global)	gitlab	ppmm98@naver.com/******
		System	(global)	EC2_SERVER_IP	EC2_SERVER_IP
		System	(global)	ssh-key	ssh-key
		System	(global)	DOCKERHUB_USER	ppmm98@naver.com/******
		System	(global)	DOCKER_REPO	ppmm98/******
		System	(global)	application-prod.yml	application-prod.yml
		System	(global)	DOCKER_FE_USER	hhhky9900@gmail.com/******
		System	(global)	DOCKER_FE_REPO	seryoii/******
		System	(global)	REACT_ENV	.env
		System	(global)	firebase-service-account.json	firebase-service-account.json

- **GitLab**: gitlab의 프로젝트를 clone 해오기위한 credential
  - `gitlab_token`: gitlab API 토큰
  - `gitlab`: gitlab ID/PW
- **ssh-key**: jenkins에서 우리의 aws ec2의 ssh에 접속하기 위한 credential
- **EC2 Server IP:** pipeline에서 EC2 Server IP를 감추기 위한 credential
  - `EC2_SERVER_IP`: 서버 주소
- **Docker Hub:** Docker hub에 있는 이미지를 끌어오기 위함
  - `DOCKER_USER`, `DOCKER_FE_USER`: Docker hub 아이디 / 비밀번호
  - `DOCKER_REPO`, `DOCKER_FE_REPO`: Docker hub nameSpace / Docker hub RepositoryName
- **백엔드 프론트엔드 설정파일들**

  프로젝트 최종 배포시 중요한 정보들이 들어있는 Spring, React 설정 파일들을 gitlab에 올리지않기 때문에 Jenkins에 미리 저장 해두고 파이프라인 속 build 전 단계에 가져오기위함
  - `REACT_ENV`: 프론트엔드 env파일
  - `application-prod.yml`: 백엔드 SpringBoot yml파일
  - `firebase-service-account.json`: firebase 설정파일

Gitlab 웹훅 설정

- 백엔드: backend 브랜치
- 프론트: frontend 브랜치

젠킨스 플러그인 추가 설치

- Gitlab
- SSH Agent
- Pipeline Graph View
- Mattermost Notification
- react

4. 배포 위한 파일 생성

[Backend - Spring]

1. **SpringBoot Dockerfile 생성**

   🖨️ **Dockerfile**

```
# open jdk 17 버전의 환경 구성
FROM openjdk:17-alpine

# tzdata 패키지 설치 및 타임존 설정
RUN ln -snf /usr/share/zoneinfo/Asia/Seoul /etc/localtime && echo Asia/Seoul > /etc/timezone

# build가 되는 시점에 JAR_FILE 경로에 jar파일 생성
ARG JAR_FILE=/build/libs/pawly-0.0.1-SNAPSHOT.jar

COPY ${JAR_FILE} /pawlyspring.jar

# 운영 및 개발에서 사용되는 환경 설정을 분리
ENTRYPOINT ["java","-jar","-Dspring.profiles.active=prod", "-Duser.timezone=Asia/Seoul", "/pa
```

2. **DockerHub에 올린 이미지를 가져와 docker compose로 서버 띄우기**

`$ vi /home/ubuntu/docker-compose.pawly8081.yml`

🖨️ **docker-compose.pawly8081.yml**

```
services:
  api:
    image: ppmm98/pawly-spring:latest
    container_name: pawly-8081
    env_file:
      - .env
    environment:
      - TZ=Asia/Seoul
      - LANG=ko_KR.UTF-8
      - HTTP_PORT=8081
    ports:
      - '8081:8080'
```

`$ vi /home/ubuntu/docker-compose.pawly8082.yml`

🖨️ **docker-compose.pawly8082.yml**

```
services:
  api:
    image: ppmm98/pawly-spring:latest
    container_name: pawly-8082
    env_file:
      - .env
    environment:
      - TZ=Asia/Seoul
      - LANG=ko_KR.UTF-8
      - HTTP_PORT=8082
    ports:
      - '8082:8080'
```

3. **BLUE/GREEN 무중단 배포 script 작성**

`$ vi /home/ubuntu/deploy.sh`

🖨️ **deploy.sh**

: EC2환경에서 배포하기 위한 스크립트

```
DOCKER_APP_NAME=pawly
# 0
# 이미지 갱신
sudo docker compose -p ${DOCKER_APP_NAME}-8081 -f /home/ubuntu/docker-compose.${DOCKER_APP_NA
sudo docker compose -p ${DOCKER_APP_NAME}-8082 -f /home/ubuntu/docker-compose.${DOCKER_APP_NA

# 1 현재 떠 있는 컨테이너 체크
EXIST_8081=$(sudo docker compose -p ${DOCKER_APP_NAME}-8081 -f /home/ubuntu/docker-compose.$
EXIST_8082=$(sudo docker compose -p ${DOCKER_APP_NAME}-8082 -f /home/ubuntu/docker-compose.$

# 2 컨테이너 스위칭
if [ -n "$EXIST_8082" ]; then
    echo "8081 컨테이너 실행"
    sudo docker compose -p ${DOCKER_APP_NAME}-8081 -f /home/ubuntu/docker-compose.${DOCKER_AP
    BEFORE_COLOR="8082"
    AFTER_COLOR="8081"
    BEFORE_PORT=8082
    AFTER_PORT=8081
else
```

```
    echo "8082 컨테이너 실행"
    sudo docker compose -p ${DOCKER_APP_NAME}-8082 -f /home/ubuntu/docker-compose.${DOCKER_AP
    BEFORE_COLOR="8081"
    AFTER_COLOR="8082"
    BEFORE_PORT=8081
    AFTER_PORT=8082
fi

# 3 서버 상태 체크
SERVER_OK=false
for cnt in `seq 1 10`; do
    echo "서버 응답 확인 : (${cnt}/10)"
    UP=$(curl -s http://127.0.0.1:${AFTER_PORT}/api/server-check)
    if [ "${UP}" = "OK" ]; then
        SERVER_OK=true
        break
    fi
    sleep 10
done


if [ "$SERVER_OK" = true ]; then
    echo "${AFTER_COLOR} server up(port:${AFTER_PORT})"

    # 4 nginx 설정 변경사항 reload
    sudo sed -i "s/${BEFORE_PORT}/${AFTER_PORT}/" /etc/nginx/conf.d/service-url.inc
    sudo nginx -s reload
    echo "Nginx reload"

    # 5 새로운 컨테이너가 제대로 떴는지 재확인
    EXIST_AFTER=$(docker compose -p ${DOCKER_APP_NAME}-${AFTER_PORT} -f docker-compose.${DOCK
    if [ -n "$EXIST_AFTER" ]; then
        # 6 이전 컨테이너 종료
        echo "$BEFORE_COLOR server down(port:${BEFORE_PORT})"
        docker compose -p ${DOCKER_APP_NAME}-${BEFORE_PORT} -f docker-compose.${DOCKER_APP_NA

        # 7 사용되지 않는 이미지 삭제
        sudo docker image prune -f
    else
        echo "새 컨테이너 실행 실패. 이전 상태로 롤백합니다."
        docker compose -p ${DOCKER_APP_NAME}-${AFTER_PORT} -f docker-compose.${DOCKER_APP_NAM
    fi
else
    echo "서버에 문제가 있어요. 배포를 중단하고 이전 상태를 유지합니다."
    # 새로 시작한 컨테이너 종료
    docker compose -p ${DOCKER_APP_NAME}-${AFTER_PORT} -f docker-compose.${DOCKER_APP_NAME}$
fi
```

새로 배포한 버전에 이상이 없으면 새로운 컨테이너로 교체, 이상이 있으면 기존 컨테이너 유지함

[Backend - Flask]

이미지 생성, 배경 제거용 서버

1. 🖨️ **Dockerfile**

```
# Python 이미지를 베이스로 사용
FROM python:3.9-slim
```

```
# 작업 디렉토리 설정
WORKDIR /app

# 필요한 파일 복사
COPY ./app /app
COPY requirements.txt /app

# 패키지 설치
RUN pip install --no-cache-dir -r requirements.txt

# Flask 애플리케이션 실행
CMD ["python", "/app/app.py"]
```

2. **docker-compose.yml 파일 생성**

`$ vi /home/ubuntu/docker-compose.pawly5000.yml`

🖨️ **docker-compose.pawly5000.yml**

```
version: '3.8'

services:
  flask-app:
    image: ppmm98/flask:latest
    container_name: flask-app-container
    env_file:
      - .flask_config.env   # 변경된 파일 이름
    environment:
      - TZ=Asia/Seoul
      - LANG=C.UTF-8
      - FLASK_ENV=production
    ports:
      - "5000:5000"
    restart: always
```

[FrontEnd]

1. **Dockerfile 생성**

🖨️ **Dockerfile (프론트엔드 프로젝트 내부)**

```
# Node.js 20 버전 이미지 기반 새로운 이미지 생성
FROM node:20

# 컨테이너 내 작업할 디렉토리 설정
WORKDIR /app

# package.json, package-lock.json 컨테이너에 복사
COPY package*.json ./

RUN rm -rf node_modules

# 의존성 설치
RUN npm ci

# 나머지 파일 컨테이너에 복사
COPY . .

# 빌드 실행
RUN npm run build
```

```
CMD ["npm", "run", "start"]
```

2. **docker-compose.yml 파일 생성**

   `$ vi /home/ubuntu/docker-compose.pawly5173.yml`

   🖨️ **docker-compose.pawly5173.yml**

```
services:
     api:
         image: seryoii/pawly-frontend:latest
         container_name: pawly-5173
         environment:
                 - TZ=Asia/Seoul
                 - LANG=ko_KR.UTF-8
                 - HTTP_PORT=5173
          ports:
                 - '5173:5173'
          command: npm run dev
```

3. **배포 script 작성**

   `$ vi /home/ubuntu/deploy-frontend.sh`

   🖨️ **deploy-frontend.sh**

```
# image 갱신
sudo docker compose -p pawly-5173 -f /home/ubuntu/docker-compose.pawly5173.yml pull

sudo docker compose -p pawly-5173 -f /home/ubuntu/docker-compose.pawly5173.yml up -d --force-

sudo docker image prune -f
```

5. 참고: EC2내 파일구조

```
$ pwd
/home/ubuntu
$ tree
.
├── 📁 Dockerfiles
├── deploy-frontend.sh
├── deploy.sh
├── docker-compose.pawly5000.yml
├── docker-compose.pawly5173.yml
├── docker-compose.pawly8081.yml
├── docker-compose.pawly8082.yml
└── 📁 jenkins-backup
```