포팅 메뉴얼

🛠️기술 스택 & 버전 정보

1. 이슈관리

2. 형상관리

3. 커뮤니케이션

4. 개발 환경

5. 상세

DB

CI/CD

Backend

Frontend

DATA

Server

UI/UX

⚙️빌드 방법

⬛BE

1. `back` 프로젝트 열기
2. JDK 17 버전 확인
3. src/main/BackApplication class 실행

ENV

```
DB_NAME=
DB_USER=
DB_PASSWORD=
DB_HOST=
DB_PORT=5432

S3_NAME=cocoa-bucket-sygy
S3_REGION=ap-northeast-2
S3_ACCESS_KEY=
S3_SECRET_KEY=
S3_URL=https://[버킷 이름].s3.ap-northeast-2.amazonaws.com

ELASTIC_HOST=elastic
ELASTIC_PORT=9200
ELASTIC_USERNAME=elastic
ELASTIC_PASSWORD=

REDIS_HOST=redis
REDIS_PORT=6379
REDIS_PASSWORD=

OAUTH2_KAKAO_CLIENT_ID=
OAUTH2_KAKAO_CLIENT_SECRET=

SERVER_BASE_URL=https://j12a507.p.ssafy.io/api
CLIENT_BASE_URL=https://j12a507.p.ssafy.io

SPRING_JWT_SECRET=
```

```
SPRING_ACCESSTOKEN_EXPIRES_IN=1800000
SPRING_REFRESHTOKEN_EXPIRES_IN=1800000
```

⬜FE

1. `front` 로 이동
2. npm install
3. npm run build
4. npm run start

ENV

```
NEXT_PUBLIC_API_BASE_URL=https://j12a507.p.ssafy.io/api

NEXT_PUBLIC_S3_URL=https://[버킷 이름].s3.ap-northeast-2.amazonaws.com
```

⬛MONITORING

1. DBeaver를 설치하고 실행
2. PostgreSQL 플러그인 설치 여부 확인
3. RDS 인스턴스 엔드포인트 및 포트 입력
    - Host: j12a507-db.******.rds.amazonaws.com
    - Port: 5432
    - DB: cocoa
    - Username / Password 입력
4. 연결 테스트 후 접속 완료
5. 실시간 테이블 상태, 쿼리 로그, 인덱스 등을 시각적으로 모니터링

🌞Docker & Jenkins (서버 배포)

1. 서버 접속

```
# Window Terminal 사용
## - ssafy 제공 서버 : Front, Back 서버 ( pem 키 사용 )
ssh -i [키명.pem] ubuntu@[탄련적IP/도메인 주소]

# 키 페어 권한이 너무 열려 있어서 접속이 안 될 때, 키 페어가 있는 위치로 이동해서
chmod 600 [Key-Pair-Name].pem
(사용자(Owner)에게만 읽고 쓰기 권한을 주고, 나머지 group과 others에 대한 권한은 전부 삭제)
chmod 400 [Key-Pair-Name].pem
```

2. 기본 설정

```
# 방화벽 설정
sudo ufw allow 22 # ssh
sudo ufw allow 80 # http
sudo ufw allow 443 # ssl
sudo ufw allow [8080/3000/9090/...] # 필요한 포트 번호 개방
sudo ufw enable

# 상태 확인
sudo ufw status

# 서울로 시간대 변경
sudo ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime
```

3. 필수 패키지 및 도커 설치

```
# 필수 패키지 설치
sudo apt update && sudo apt upgrade -y

# 도커 설치
sudo apt install -y docker.io

# 도커에 사용자 권한 추가
sudo usermod -aG docker $USER
newgrp docker
```

```
# 도커 컴포즈 설치
sudo curl -L "https://github.com/docker/compose/releases/download/v2.34.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose

# 도커 컴포즈 권한 부여
sudo chmod +x /usr/local/bin/docker-compose

# 심볼릭 링크 설정(설정 안 하면 path 에러 발생)
sudo ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

# nvm 설치
curl -o- https://raw.githubusercontent.com/nvm-sh/nvm/v0.39.7/install.sh | bash

# nvm 로드
source ~/.bashrc

# node.js lts 버전 설치
nvm install --lts
```

4. 젠킨스 설치

docker와 docker-compose에 접근하려고 하면 지속적으로 권한 문제로 도커 Daemon에 접근하지 못하는 문제가 생김 → 커스텀 도커 이미지로 젠킨스 컨테이너 빌드

1. 젠킨스의 docker container에 마운트할 볼륨 디렉토리 생성

   ```
   cd /home/ubuntu && mkdir jenkins-data
   ```

2. 외부에서 접속할 포트를 오픈

   ```
   sudo ufw allow 9090
   ```

3. EC2에 젠킨스 이미지를 pull

   ```
   docker image pull jenkins/jenkins:jdk1
   ```

4. jenkins 관련 파일들을 모아두는 디렉토리 생성 후 이동

```
mkdir ~/jenkins-docker
cd ~/jenkins-docker
```

5. Dockerfile작성 (생성: `nano Dockerfile`)

```
FROM jenkins/jenkins:jdk17

USER root

# docker docker-compose 설치
RUN apt-get update && apt-get install -y docker.io
RUN curl -L "https://github.com/docker/compose/releases/download/v
2.34.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/d
ocker-compose && chmod +x /usr/local/bin/docker-compose

#  기존 docker 그룹 제거하고 새로 생성
RUN groupdel docker || true && groupadd -g 122 docker

# jenkins 유저를 docker 그룹에 추가
RUN usermod -aG docker jenkins

# Node.js (LTS 버전) 설치 (공식 권장 방법)
RUN curl -fsSL https://deb.nodesource.com/setup_22.x | bash - && apt
-get install -y nodejs

USER jenkins
```

6. Docker 이미지 빌드

```
docker build -t custom-jenkins-docker .
```

7. 이미지를 활용해서 젠킨스 컨테이너 실행

5. Jenkins 환경 설정

```
# jenkins-data 폴더로 이동
cd jenkins-data
```

```
# update center에 필요한 CA 파일을 다운로드
mkdir update-center-rootCAs
wget https://cdn.jsdelivr.net/gh/lework/jenkins-update-center/rootCA/update-center.crt -O ./update-center-rootCAs/update-center.crt

# 젠킨스의 디폴트 설정에서 특정 미러사이트로 대체(필수)
sudo sed -i 's#https://updates.jenkins.io/update-center.json#https://raw.githubusercontent.com/lework/jenkins-update-center/master/updates/tencent/update-center.json#' ./hudson.model.UpdateCenter.xml

# Jenkins 서비스 구동(컨테이너 실행)
docker restart jenkins

# jenkins 로그 확인 명령어
docker logs jenkins

# jenkins 실시간 로그 확인 명령어
docker logs -f jenkins

-----
# 이미 실행중인 컨테이너에 prefix 설정하는 법
docker exec -it -u root jenkins /bin/bash

cd /var/jenkins_home

echo "JENKINS_OPTS=\"--prefix=/jenkins\"" >> jenkins.properties

------
젠킨스에 도커 권한 부여
# 젠킨스 사용자 확인
ps aux | grep jenkins

# 사용자에게 권한 부여
sudo usermod -aG docker jenkins

# 젠킨스 재시작
```

6. Jenkins config 보안 설정 확인

1. config.xml 확인

```
vi /home/ubuntu/jenkins-data/config.xml
# 명령어로 파일 열어서
<useSecurity>true</useSecurity>
<securityRealm class="hudson.security.HudsonPrivateSecurityRealm">
    <disableSignup>true</disableSignup>
# 같은 상태인지 확인
```

2. jenkins 웹에서 security 설정이 정상적으로 되어있는지 확인
    - 위치: Jenkins 관리 > Security
    - Security Realm: Jenkins' own user database
    - Authorization: Logged-in users can do anything
    - Allow anonymous read access 체크 해제

7. Jenkins 초기 설정

1. http://<EC2 도메인네임>/jenkins 접속
2. 별도로 기록해둔 로그의 초기 패스워드 입력
3. Install suggest plugins 클릭 ⇒ 기본 추천 플러그인 설치
4. 관리 계정 설정
5. 외부 접속 url 설정 : http://<EC2 도메인네임>/jenkins
6. 젠킨스에 docker 및 docker-compose 설치

```
# jenkins 컨테이너 루트로 접속
docker exec -u root -it jenkins bash

# apt 업데이트 및 docker-cli 설치
apt-get update
apt-get install -y docker.io

# 도커 컴포즈 설치
curl -L "https://github.com/docker/compose/releases/download/v2.34.0/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose
```

```
# 도커 컴포즈 권한 부여
chmod +x /usr/local/bin/docker-compose

# 도커 컴포즈 버전 확인(출력되면 잘 설치된 것)
docker-compose --version

# 버전이 출력되지 않으면 심볼릭 링크 설정
ln -s /usr/local/bin/docker-compose /usr/bin/docker-compose

# jenkins 유저를 docker 그룹에 추가
usermod -aG docker jenkins

# 권한 확인하는 법
id jenkins
```

설치 후에는 컨테이너를 재시작 해야 한다.

8. GitLab jenkins 연결(초기 연결 테스트)

1. 젠킨스에 깃랩 플러그인 설치
    a. 젠킨스 관리 > Plugins > Available plugins
    b. gitlab 플러그인 설치
2. 깃랩에서 personal access token 발급
3. 젠킨스 Credentials 등록
    a. 젠킨스 관리 > Credentials > Add credentials
    b. Username with password 등록

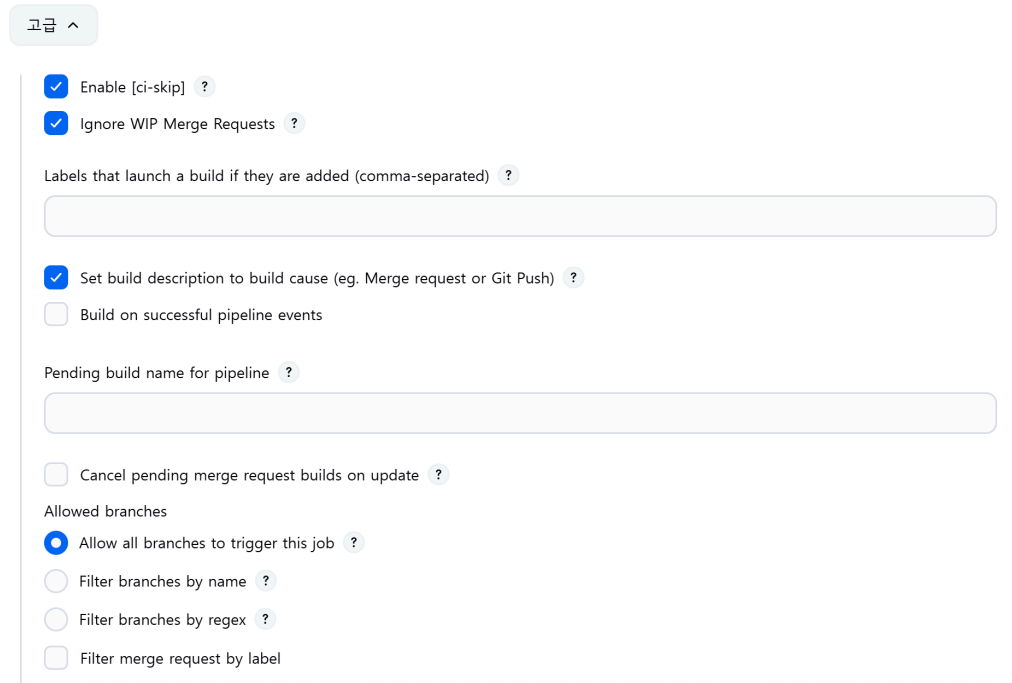
🔑

- kind : Username with password 선택
- Scope : Global (Jenkins, nodes, items, all child items, etc) 선택
- Username : GitLab 사용중인 ID
- Password : GitLab 에서 발급받은 토큰 값
- ID : 해당 Credentials를 식별하기 위한 ID (사용자 임의의 문자열로 기입)
- Description: 설명

c. GitLab API Token 등록

4. Item - 소스코드 관리 추가
   a. 새로운 Item 등록
      i. 프로젝트 이름 입력
      ii. **Freestyle project 선택**
   b. 소스코드 관리에서 Git 선택
      i. Repository URL에 프로젝트 url 입력
      ii. Credentials 등록한 username 선택
      iii. jenkins build가 돌아갈 브랜치를 지정
   c. 빌드 설정
      i. 빌드 유발에서는 GitLab 선택 후 뒤에 붙어있는 URL을 메모

ii. 고급 선택 후 secret token이 비어있으면 Generate하여 토큰 발급



iii. GitLab Webhook 등록

1. URL - 아까 메모해뒀던 URL
2. Secret token - 젠킨스에서 발급받은 secret token
3. 빌드 시 연결되기 원하는 것들에 체크
    a. develop 브랜치 push는 이렇게

4. 저장 후 테스트

9. Nginx 설치 및 설정

1. nginx 설치
   EC2에 직접 설치하였기 때문에 이식성은 낮음

```
sudo apt install nginx -y
```

2. nginx 실행

```
sudo systemctl start nginx
```

3. nginx 설정
   a. nginx.conf http 블록 안에 `include /etc/nginx/custom-nginx.conf` 추가

```
cd /etc/nginx
nano nginx.conf
http 블록 안에 include /etc/nginx/custom-nginx.conf 추가
```

   b. custom-nginx.conf 파일에 서버 설정

```
# 설정 파일 복사해서 초기화 후 서버 설정 내용 입력
cp  nginx.conf custom-nginx.conf

-----------------------------------------------------
server {
    if ($host = j12a507.p.ssafy.io) {
        return 301 https://$host$request_uri;
    } # managed by Certbot
```

```
    listen       80;
    listen  [::]:80;
    server_name  j12a507.p.ssafy.io;
    return 404; # managed by Certbot
}

server {
    listen 443 ssl;
    #listen [::]:443 ssl;
    #http2 on;
    server_name j12a507.p.ssafy.io;
    ssl_certificate /etc/letsencrypt/live/j12a507.p.ssafy.io/fullchain.pe
m;
    ssl_certificate_key /etc/letsencrypt/live/j12a507.p.ssafy.io/privke
y.pem;
    ssl_buffer_size 64k;

    # 포워딩
    location / { # front
        proxy_pass http://127.0.0.1:3000/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
or;
        proxy_set_header X-Forwarded-Proto https;
        proxy_http_version 1.1;
        proxy_set_header Connection 'keep-alive';
    }

    location /jenkins/ {
        proxy_pass http://127.0.0.1:9090/jenkins/;
        proxy_set_header Host $host;
        proxy_set_header X-Real-IP $remote_addr;
        proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
or;
        proxy_set_header X-Forwarded-Proto https;
```

```
    }

    location /api-docs/ {
        proxy_pass http://127.0.0.1:8080/api-docs/;
    }

    location /api/ {
      proxy_pass http://127.0.0.1:8080/;
      proxy_set_header Host $host;
      proxy_set_header X-Real-IP $remote_addr;
      proxy_set_header X-Forwarded-For $proxy_add_x_forwarded_f
or;
      proxy_set_header X-Forwarded-Proto https;
      proxy_set_header Connection 'keep-alive';
      proxy_http_version 1.1;
      proxy_buffering off;
      proxy_read_timeout 3600;
      proxy_send_timeout 3600;
      #chunked_transfer_encoding off;
      #proxy_set_header X-Accel-Buffering "no";
    }
}
```

4. Certbot 설치

```
sudo apt install certbot python3-certbot-nginx -y
```

5. SSL 인증서 발급
    a. 아래 코드를 입력하면 대화형 프롬프트가 나타나고, 이메일 입력하여 이용약관 동의

```
sudo certbot --nginx -d [서비스 도메인]
# 만약 다른 도메인도 설정하고 싶으면 -d [원하는도메인] 명령어에 붙이기
```

6. Certbot 자동 갱신 설정
    a. Certbot 자동 갱신 설정
        i. Certbot은 기본적으로 자동 갱신 크론 작업을 생성

```
# 확인
sudo systemctl systemctl list-timers | grep certbot
# snap.certbot.renew.timer  snap.certbot.renew.service 출력됨
```

ii. 자동갱신 안 된다면

```
sudo crontab -e
# 편집기 열리면 아랫줄에 추가
0 0 * * * /usr/bin/certbot renew --quiet
매일 자정에 인증서가 갱신
```

b. 수동 갱신 테스트

```
sudo certbot renew --dry-run
```

c. nginx 설정 파일 편집 → 443 port 관련 설정 수정이 이 단계에서 이뤄짐

d. nginx 설정 테스트 및 재시작

```
sudo nginx -t
sudo systemctl restart nginx
```

7. Jenkins & Nginx 연동

a. Nginx가 `/jenkins` 요청을 Docker 내부의 Jenkins 컨테이너로 전달하는 설정을 해줘야 함

b. custom-nginx.conf에서 jenkins 서버 설정 proxy_pass를 `127.0.0.1:9090`에서 `127.0.0.1:9090/jenkins`로 변경

c. 젠킨스 기본 URL `도메인/jenkins`로 변경 → 젠킨스 UI > Jenkins 관리 > System에서 설정
← 만약 생성 시에 잘못 생성했다면!

d. 역방향 프록시 어쩌구 뜨면, 컨테이너 내부 config.xml 수정

```
# 컨테이너에서 호스트로 복사
docker cp jenkins:/var/jenkins_home/config.xml ~/config.xml

# 아래 내용 최상위 태그 맨 위에 추가
```

```
  <jenkinsUrl>https://도메인/jenkins/</jenkinsUrl>

# 수정한 파일 호스트에서 컨테이너로 복사
docker cp ~/config.xml jenkins:/var/jenkins_home/config.xml
```

10. 각 파트 컨테이너 빌드

1. **Dockerfile 작성** : 필요한 서비스를 정의
    a. 프론트

```
FROM node:18-slim

# 작업 디렉토리 설정
WORKDIR /app

# Jenkins에서 빌드된 결과물 복사
COPY .next .next
COPY public public
COPY package.json .
COPY next.config.ts .
COPY node_modules node_modules

# 포트 개방
EXPOSE 3000

# Next.js 서버 실행
CMD ["npm", "run", "start"]
```

    b. 백

```
FROM bellsoft/liberica-openjdk-alpine:17

# TimeZone 설정
ENV TZ=Asia/Seoul
RUN apk --no-cache add tzdata

WORKDIR /usr/src/app
```

```
# 빌드 산출물(JAR) 복사
COPY build/libs/*.jar cocoa.jar
# Dockerfile이 있는 위치를 기준으로 경로를 설정해야 함

# 컨테이너에서 열 포트 (예: 8080)
EXPOSE 8080

# 컨테이너 실행 시 명령
ENTRYPOINT ["java", "-jar", "cocoa.jar"]
```

c. 젠킨스

d. 로컬에서 개별 컨테이너 실행 및 테스트

i. 터미널에서 프로젝트 폴더로 이동

ii. 도커 이미지 빌드 : `docker build -t my-nextjs-test .`

iii. 도커 컨테이너 실행 : `docker run -p 3000:3000 --name nextjs-container my-nextjs-test`

iv. Docker Desktop에서 컨테이너 확인 가능

v. 컨테이너 정지 : `docker stop nextjs-container`

vi. 컨테이터 삭제 : `docker rm nextjs-container`

vii. 이미지 삭제 : `docker rmi my-nextjs-test`

2. **docker-compose.yml 작성** : 각 서비스 간의 관계를 정의

a. 브랜치 루에 docker-compose.yml 생성

b. docker-compose.yml 작성

```
services:
  back:
    build: ./back
    container_name: back
    ports:
      - "8080:8080"
    restart: always
    env_file:
      - .env
    environment:
      TZ: Asia/Seoul
```

```
    depends_on:
      redis:
        condition: service_healthy
      elastic:
        condition: service_healthy
    networks:
      - cocoa-net

front:
    build: ./front
    container_name: front
    ports:
      - "3000:3000"
    restart: always
    env_file:
      - .env
    environment:
      TZ: Asia/Seoul
    depends_on:
      - back
    networks:
      - cocoa-net

redis:
    image: redis:7-alpine
    container_name: redis
    healthcheck:
  test: ["CMD", "redis-cli", "-a", "${REDIS_PASSWORD}", "ping"]
  interval: 3s
  retries: 5
  start_period: 1s
  timeout: 10s
    restart: always
    command:
          ["redis-server",
          "--requirepass", "${REDIS_PASSWORD}",
          "maxmemory", "256mb",
          "maxmemory-policy", "volatile-lru"]
```

```
    environment:
      TZ: Asia/Seoul
      REDIS_PASSWORD: ${REDIS_PASSWORD}
    expose:
      - "6379"
    volumes:
    - redis-data:/data
    networks:
      - cocoa-net

  elastic:
    image: docker.elastic.co/elasticsearch/elasticsearch:8.17.3
    container_name: elastic
    healthcheck:
   test: ["CMD", "curl", "-f", "http://localhost:9200"]
   interval: 5s
   retries: 60
   start_period: 5s
   timeout: 10s
    restart: always
    environment:
      discovery.type: single-node
      TZ: Asia/Seoul
      ES_JAVA_OPTS: "-Xms1g -Xmx1g"
      bootstrap.memory_lock: true
      xpack.security.enabled: false
      xpack.security.http.ssl.enabled: false
      xpack.security.transport.ssl.enabled: false
    expose:
      - "9200"
    volumes:
    - elastic-data:/usr/share/elasticsearch/data
    networks:
      - cocoa-net

networks:
  cocoa-net:
    driver: bridge
```

```
volumes:
  redis-data:
  elastic-data:
```

3. **로컬에서 컨테이너 실행 및 테스트**
   a. Docker Desktop 실행 후 컨테이너 빌드 & 실행

```
docker compose up --build -d
```

   b. docker ps로 컨테이너 빌드 되었는지 확인
   c. docker compose down로 로컬에서 실행 중지
   d. redis와 elastic search에 사용되는 포트 개방할 것
4. **GitLab에 푸시**

11. Jenkins pipeline 구축

1. Jenkins 관리 > Plugins > Pipeline 플러그 설치 확인
2. 새로운 Item 생성 > Pipeline으로 선택
3. Triggers에서 Build when a change is pushed to GitLab 선택
   a. 웹훅과 시크릿 토큰 별도 저장해서 깃랩 웹훅에 사용
4. Jenkinsfile에 파이프라인 코드 작성
   a. docker-compose를 통해서 빌드하는 파이프라인 스크립트

```
pipeline {
  agent any

  environment {
    ENV_FILE = credentials('env')
  }

  stages {
   stage('Checkout') {
   steps {
```

```
            git branch: 'develop',
            credentialsId: 'GitLab_personal_token',
            url: 'https://lab.ssafy.com/s12-bigdata-dist-sub1/S12P21A507.g
    }
    }

    stage('Prepare .env') {
        steps {
        dir('back') {
          sh 'rm -f .env'
        sh 'cp $ENV_FILE .env'
      }
      dir('front') {
        sh 'rm -f .env'
        sh 'cp $ENV_FILE .env'
      }
      sh 'cp $ENV_FILE ./env_temp && mv ./env_temp .env
        }
    }

    stage('Test Backend') {
    steps {
      dir('back') {
          sh 'chmod +x ./gradlew'
        sh './gradlew test'
      }
    }
  }

  stage('Build Backend') {
    steps {
      dir('back') {
        sh './gradlew clean build -x test'
      }
    }
  }

  stage('Build Frontend') {
```

```
  steps {
    dir('front') {
       sh 'npm install'
      sh 'npm run build'
    }
  }
}

  stage('Docker Compose back front Down') {
  steps {
       sh 'docker-compose stop back front || true'
    sh 'docker-compose rm -f back front || true'
  }
}

  stage('Docker Compose back front Build') {
    steps {
    sh 'docker-compose build back front'
  }
  }

  stage('Docker Compose back front Up') {
  steps {
    sh 'docker-compose up -d back front'
  }
}
}

post { /
    always {
       echo '🧹 Docker cleanup 시작...'
       sh '''
          docker image prune -f
          docker builder prune -f
       '''
    }
success {
  echo '🎉 배포 성공!'
```

```
      mattermostSend(
        color: "good",
        message: "🎉 배포 성공! Job: ${env.JOB_NAME}, Build: #${env.BU
      )
    }
    failure {
      echo '🚨 배포 실패!'
      mattermostSend(
          color: "danger",
        message: "🚨 배포 실패! Job: ${env.JOB_NAME}, Build: #${env.BU
      )
    }
  }
}
```

💻MONITORING (DBeaver로 RDS PostgreSQL 모니터링)

1. DBeaver 설치
    - 공식 사이트: https://dbeaver.io/
2. DBeaver 실행 후 "Database > New Database Connection" 선택
3. PostgreSQL 선택 후 RDS 접속 정보 입력
    - Host: [RDS Endpoint] (예: j12a507-db.***.rds.amazonaws.com)
    - Port: 5432
    - Database: [DB 이름] (예: cocoa)
    - Username: [username]
    - Password: [비밀번호]
4. "Test Connection" 클릭 → 성공 시 "Finish"
5. 연결 후 좌측 탐색기에서:
    - 테이블 스키마, 레코드 수, 인덱스 상태 확인
    - 실시간 SQL 로그/쿼리 실행 결과 모니터링 가