빌드 및 배포 환경

DockerFIle

- FrontEnd

```
FROM node:20-alpine AS build

WORKDIR /app
COPY package*.json ./
RUN npm install --legacy-peer-deps
COPY . .
# 타입 검사 없이 빌드 실행
RUN NEXT_IGNORE_TS_ERRORS=1 npm run build

# 프로덕션 환경에서는 dev dependencies 제외
FROM node:20-alpine AS production

# 타임존 설정
RUN apk add --no-cache tzdata && \
    ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime && \
    echo "Asia/Seoul" > /etc/timezone

WORKDIR /app
COPY --from=build /app/package*.json ./
RUN npm ci --only=production --legacy-peer-deps
COPY --from=build /app/.next ./.next
COPY --from=build /app/public ./public

COPY --from=build /app/next.config.* ./

EXPOSE 3000
CMD ["npm", "run", "start"]
```

- BackEnd

```
# 1. OpenJDK 이미지 사용
FROM openjdk:17-jdk-alpine AS build

# 2. 작업 디렉토리 설정
WORKDIR /app

# 3. 필요한 파일 복사 (전체가 아닌 필요한 부분만)
COPY ./src ./src
COPY ./build.gradle ./build.gradle
COPY ./settings.gradle ./settings.gradle
COPY ./gradlew ./gradlew
COPY ./gradle ./gradle

# 4. Gradle 파일 실행 권한 부여 (Linux 환경에서 필요)
RUN chmod +x ./gradlew

# 5. Gradle 빌드 실행 (테스트 제외)
RUN ./gradlew build -x test --no-daemon

# 6. 런타임 이미지 생성
FROM openjdk:17-jdk-alpine

# 타임존 설정 추가
RUN apk add --no-cache tzdata && \
    ln -sf /usr/share/zoneinfo/Asia/Seoul /etc/localtime && \
    echo "Asia/Seoul" > /etc/timezone

# Java 애플리케이션 타임존 설정
ENV JAVA_OPTS="-Duser.timezone=Asia/Seoul"

WORKDIR /app
COPY --from=build /app/build/libs/*.jar app.jar

# 7. 실행 명령어 설정
EXPOSE 8080
ENTRYPOINT ["java", "-jar", "app.jar"]
```

- nginx

```
FROM nginx:latest
COPY conf.d/default.conf /etc/nginx/conf.d/default.conf
```

Docker Compose

- canary, stable 각 파일은 컨테이너 이름만 상이하며 내용은 동일함.

```
services:

  client-canary:
    build:
      context: ./client
      dockerfile: Dockerfile
    container_name: client-canary
    ports:
      - "3002:3000"
    depends_on:
      - server-canary
    environment:
      - NODE_ENV=production
      - NEXT_PUBLIC_API_URL=/api
      - TZ=Asia/Seoul
    restart: always
    networks:
      - app-network

  server-canary:
    build:
      context: ./server
      dockerfile: Dockerfile
    container_name: server-canary
    ports:
      - "8082:8080"
    environment:
      - SPRING_PROFILES_ACTIVE=${SPRING_PROFILES}
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
```

```
      - MYSQL_DATABASE=${MYSQL_DATABASE}
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}
      - SPRING_DATASOURCE_URL=jdbc:mysql://my-db:3306/${MYSQL_DA
TABASE}?useSSL=false&serverTimezone=Asia/Seoul&characterEncoding=
UTF-8&allowPublicKeyRetrieval=true
      - SPRING_DATASOURCE_USERNAME=${MYSQL_USER}
      - SPRING_DATASOURCE_PASSWORD=${MYSQL_PASSWORD}
      - JWT_SECRET=${JWT_SECRET}
      - JWT_ACCESS_TOKEN_VALIDITY=${JWT_ACCESS_TOKEN_VALIDITY}
      - JWT_REFRESH_TOKEN_VALIDITY=${JWT_REFRESH_TOKEN_VALIDIT
Y}
      - MY_DB=${MY_DB}
      - SPRING_DATA_REDIS_HOST=${SPRING_DATA_REDIS_HOST}
      - SPRING_DATA_REDIS_PORT=${SPRING_DATA_REDIS_PORT}
      - SPRING_DATA_REDIS_PASSWORD=${SPRING_DATA_REDIS_PASSWO
RD}
      # AWS S3 환경 변수 추가
      - CLOUD_AWS_CREDENTIALS_ACCESS_KEY=${CLOUD_AWS_CREDEN
TIALS_ACCESS_KEY}
      - CLOUD_AWS_CREDENTIALS_SECRET_KEY=${CLOUD_AWS_CREDENT
IALS_SECRET_KEY}
      - CLOUD_AWS_S3_BUCKET=${CLOUD_AWS_S3_BUCKET}
      - CLOUD_AWS_REGION_STATIC=ap-northeast-2
      - CLOUD_AWS_STACK_AUTO=false
      - CLAUDE_APIKEY=${CLAUDE_APIKEY}
      - TZ=Asia/Seoul
    # depends_on:
    #   my-db:
    #     condition: service_healthy
    #   my-cache-server:
    #     condition: service_healthy
    networks:
      - app-network
    restart: always

  python-canary:
    build:
```

```
      context: ./python-server
      dockerfile: Dockerfile
    container_name: python-canary
    ports:
      - "8020:8000"
    environment:
      - DEBUG=False
      - MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PASSWORD}
      - MYSQL_DATABASE=${MYSQL_DATABASE}
      - MYSQL_USER=${MYSQL_USER}
      - MYSQL_PASSWORD=${MYSQL_PASSWORD}
      - JWT_SECRET=${JWT_SECRET}
      - CLAUDE_APIKEY=${CLAUDE_APIKEY}
      - TZ=Asia/Seoul
    volumes:
      - ./python-server:/app
    networks:
      - app-network
    restart: always

networks:
  app-network:
    external: true
```

Jenkins 파이프라인

```
// 한국 시간으로 날짜 포맷팅하는 함수
def getKoreanTime() {
    def now = new Date()
    def koreanTimeZone = TimeZone.getTimeZone("Asia/Seoul")
    def sdf = new java.text.SimpleDateFormat("yyyy-MM-dd HH:mm:ss")
    sdf.setTimeZone(koreanTimeZone)
    return sdf.format(now)
}

// 현재 Nginx 설정에서 가중치 확인 함수
```

```
def getCurrentWeights() {
    def nginxConf = ""

    sshagent(['ssh-ec2-ubuntu']) {
        // 설정 확인 디버깅 출력
        sh """
            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                echo "현재 Nginx 설정 확인:"
                cat /home/ubuntu/project/S12P31B103/nginx/conf.d/default.conf |
grep -A 2 "upstream"
                echo "가중치 설정 확인:"
                cat /home/ubuntu/project/S12P31B103/nginx/conf.d/default.conf |
grep "weight"
            '
        """

        // 설정 파일 내용 가져오기
        nginxConf = sh(script: """
            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                cat /home/ubuntu/project/S12P31B103/nginx/conf.d/default.conf
            '
        """, returnStdout: true).trim()
    }

    // 백엔드(서버) 가중치 확인
    def stableServerWeight = 0
    def canaryServerWeight = 0

    def stableServerMatch = nginxConf =~ /server server-stable:8080 weig
ht=(\d+);/
    def canaryServerMatch = nginxConf =~ /server server-canary:8080 wei
ght=(\d+);/

    if (stableServerMatch) {
        stableServerWeight = stableServerMatch[0][1].toInteger()
    }

    if (canaryServerMatch) {
```

```
        canaryServerWeight = canaryServerMatch[0][1].toInteger()
    }

    // 프론트엔드(클라이언트) 가중치 확인
    def stableClientWeight = 0
    def canaryClientWeight = 0

    def stableClientMatch = nginxConf =~ /server client-stable:3000 weight
=(\d+);/
    def canaryClientMatch = nginxConf =~ /server client-canary:3000 weigh
t=(\d+);/

    if (stableClientMatch) {
        stableClientWeight = stableClientMatch[0][1].toInteger()
    }

    if (canaryClientMatch) {
        canaryClientWeight = canaryClientMatch[0][1].toInteger()
    }

    // Python 서비스 가중치 확인
    def stablePythonWeight = 0
    def canaryPythonWeight = 0

    def stablePythonMatch = nginxConf =~ /server python-stable:8000 weig
ht=(\d+);/
    def canaryPythonMatch = nginxConf =~ /server python-canary:8000 we
ight=(\d+);/

    if (stablePythonMatch) {
        stablePythonWeight = stablePythonMatch[0][1].toInteger()
    }

    if (canaryPythonMatch) {
        canaryPythonWeight = canaryPythonMatch[0][1].toInteger()
    }

    echo "확인된 가중치: stableServer=${stableServerWeight}, canaryServer=
```

```
${canaryServerWeight}, stableClient=${stableClientWeight}, canaryClient=
${canaryClientWeight}, stablePython=${stablePythonWeight}, canaryPytho
n=${canaryPythonWeight}"

    return [
        stableServer: stableServerWeight,
        canaryServer: canaryServerWeight,
        stableClient: stableClientWeight,
        canaryClient: canaryClientWeight,
        stablePython: stablePythonWeight,
        canaryPython: canaryPythonWeight
    ]
}

// Nginx 설정 업데이트 함수 - 가중치 설정 (전체 파일 교체 방식)
def updateNginxWeights(stablePercentage, canaryPercentage) {
    echo "Nginx 가중치 업데이트 시작: Stable=${stablePercentage}%, Canary=
${canaryPercentage}%"

    //  정수 변환 (int 캐스팅)
    def stableWeight = (stablePercentage / 10) as int
    def canaryWeight = (canaryPercentage / 10) as int

    // weight=0인 경우 최소값 1로 설정
    if (canaryWeight == 0) canaryWeight = 1
    if (stableWeight == 0) stableWeight = 1

    // 비율 계산 (canaryWeight가 1이고 stableWeight가 99인 경우)
    if (canaryPercentage == 0) {
        stableWeight = 999
        canaryWeight = 1
    } else if (stablePercentage == 0) {
        canaryWeight = 999
        stableWeight = 1
    }

    sshagent(['ssh-ec2-ubuntu']) {
        // 현재 설정 파일 백업
```

```
        sh """
            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                cp /home/ubuntu/project/S12P31B103/nginx/conf.d/default.conf /
home/ubuntu/project/S12P31B103/nginx/conf.d/default.conf.bak
                echo "현재 Nginx 설정 백업 완료"
            '
        """

        // 백슬래시와 달러 기호를 처리하는 정확한 방법으로 코드 수정
        def nginxConfig = """# Canary 배포를 위한 백엔드 서비스 구성 (Spring)
upstream canary_server {
    server server-stable:8080 weight=${stableWeight};
    server server-canary:8080 weight=${canaryWeight};
}

# Canary 배포를 위한 프론트엔드 서비스 구성 (Next.js)
upstream canary_client {
    server client-stable:3000 weight=${stableWeight};
    server client-canary:3000 weight=${canaryWeight};
}

# Canary 배포를 위한 Python 서비스 구성
upstream canary_python {
    server python-stable:8000 weight=${stableWeight};
    server python-canary:8000 weight=${canaryWeight};
}

# HTTPS 설정
server {
    listen 443 ssl;
    server_name k12b103.p.ssafy.io;

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/k12b103.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k12b103.p.ssafy.io/privkey.pem;

    # SSL 보안 설정
    ssl_protocols TLSv1.2 TLSv1.3;
```

```
    ssl_prefer_server_ciphers off;
    ssl_ciphers 'HIGH:!aNULL:!MD5';

    # API 요청 처리
    location /api {
        proxy_pass http://canary_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade \${DOLLAR}http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host \${DOLLAR}host;
        proxy_set_header X-Real-IP \${DOLLAR}remote_addr;
        proxy_set_header X-Forwarded-For \${DOLLAR}proxy_add_x_forwarde
d_for;
        proxy_set_header X-Forwarded-Proto \${DOLLAR}scheme;
        # 타임아웃 설정
        proxy_connect_timeout 300;
        proxy_send_timeout 300;
        proxy_read_timeout 300;
    }

    # 중복된 /api/api 경로 처리
    location /api/api/ {
        # 첫 번째 /api/를 제거
        rewrite ^/api/api/(.*) /api/\${DOLLAR}1 break;
        proxy_pass http://canary_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade \${DOLLAR}http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host \${DOLLAR}host;
        proxy_set_header X-Real-IP \${DOLLAR}remote_addr;
        proxy_set_header X-Forwarded-For \${DOLLAR}proxy_add_x_forwarde
d_for;
        proxy_set_header X-Forwarded-Proto \${DOLLAR}scheme;
    }

    # Python API 요청 처리
    location /api/python/ {
        proxy_pass http://canary_python;
```

```
        proxy_http_version 1.1;
        proxy_set_header Upgrade \${DOLLAR}http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host \${DOLLAR}host;
        proxy_set_header X-Real-IP \${DOLLAR}remote_addr;
        proxy_set_header X-Forwarded-For \${DOLLAR}proxy_add_x_forwarde
d_for;
        proxy_set_header X-Forwarded-Proto \${DOLLAR}scheme;
    }

    # 웹소켓 라우팅 설정
    location /ws {
        proxy_pass http://canary_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade \${DOLLAR}http_upgrade;
        proxy_set_header Connection "upgrade";
        proxy_set_header Host \${DOLLAR}host;
        proxy_set_header X-Real-IP \${DOLLAR}remote_addr;
        proxy_set_header X-Forwarded-For \${DOLLAR}proxy_add_x_forwarde
d_for;
        proxy_set_header X-Forwarded-Proto \${DOLLAR}scheme;

        proxy_read_timeout 3600s;
        proxy_send_timeout 3600s;
        proxy_buffering off;
    }

    # Next.js의 정적 파일 캐싱
    location /_next/static/ {
        proxy_pass http://canary_client/_next/static/;
        proxy_set_header Host \${DOLLAR}host;
        expires 30d;
        add_header Cache-Control "public, max-age=2592000";
    }

    # 프론트엔드 라우팅
    location / {
        proxy_pass http://canary_client;
```

```
        proxy_http_version 1.1;
        proxy_set_header Upgrade \${DOLLAR}http_upgrade;
        proxy_set_header Connection 'upgrade';
        proxy_set_header Host \${DOLLAR}host;
        proxy_set_header X-Real-IP \${DOLLAR}remote_addr;
        proxy_set_header X-Forwarded-For \${DOLLAR}proxy_add_x_forwarde
d_for;
        proxy_set_header X-Forwarded-Proto \${DOLLAR}scheme;
    }
}

# HTTP → HTTPS 리다이렉션
server {
    listen 80;
    server_name k12b103.p.ssafy.io;
    return 301 https://\${DOLLAR}host\${DOLLAR}request_uri;
}"""

        // 컨텍스트 변수로 DOLLAR를 정의하여 $ 문자를 안전하게 처리
        def sshResult = sh(script: """
            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                # 달러 기호 변수 정의
                DOLLAR='\$'

                # 설정 파일 생성
                cat > /home/ubuntu/project/S12P31B103/nginx/conf.d/default.con
f << 'EOF'
${nginxConfig}
EOF

                echo "설정 파일 업데이트 완료, 내용 확인:"
                cat /home/ubuntu/project/S12P31B103/nginx/conf.d/default.conf |
grep "weight"

                # Nginx 컨테이너 재시작
                docker restart nginx

                # 잠시 대기
```

```
                sleep 3

                # 컨테이너 상태 확인
                echo "Nginx 컨테이너 상태:"
                docker ps | grep nginx
            '
        """, returnStatus: true)

        if (sshResult == 0) {
            echo "Nginx 가중치가 성공적으로 업데이트되었습니다! Stable: ${stableP
ercentage}% (${stableWeight}), Canary: ${canaryPercentage}% (${canary
Weight})"
            return true
        } else {
            echo "Nginx 가중치 업데이트 중 문제가 발생했습니다."
            error "Nginx 설정 적용 실패"
            return false
        }
    }
}

// 자동화된 모니터링 및 롤백 결정 함수
def monitorCanaryDeployment(canaryWeight) {
    echo "Canary 배포 모니터링 (${canaryWeight}% 트래픽)..."

    try {
        def canaryHealth = ""
        def stableHealth = ""

        sshagent(['ssh-ec2-ubuntu']) {
            canaryHealth = sh(script: """
                ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                    curl -s http://localhost:8082/api/health
                '
            """, returnStdout: true).trim()

            stableHealth = sh(script: """
                ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
```

```
                curl -s http://localhost:8081/api/health
            '
        """, returnStdout: true).trim()
    }

    echo "Canary 서버 헬스 상태: ${canaryHealth}"
    echo "Stable 서버 헬스 상태: ${stableHealth}"

    return canaryHealth.contains("UP") || canaryHealth.contains("OK")
  } catch (Exception e) {
    echo "모니터링 중 오류 발생: ${e.message}"
    return false
  }
}


// 서버 상태 체크 함수
def checkServerHealth(server, port) {
  def maxRetries = 3
  def retryInterval = 5
  def healthURL = "/api/health"  // 기본 경로

  // 파이썬 서버인 경우 다른 경로 사용
  if (server.contains("python")) {
    healthURL = "/api/python/health"
  }

  echo "서버 ${server} 헬스체크 시작 (URL: http://localhost:${port}${healthU
RL})..."

  for (int i = 0; i < maxRetries; i++) {
    try {
      def response = ""
      sshagent(['ssh-ec2-ubuntu']) {
        response = sh(script: """
          ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
            curl -s -m 10 http://localhost:${port}${healthURL}
          '
```

```
                """, returnStdout: true).trim()
            }

            echo "헬스체크 응답: ${response}"

            if (response && (response.contains("UP") || response.contains("ok")
|| response.contains("OK") || response.contains("status"))) {
                echo "서버 헬스체크 성공: ${server} (포트: ${port})"
                return true
            }

            if (i < maxRetries - 1) {
                echo "재시도 중... (${i+1}/${maxRetries})"
                sleep(time: retryInterval, unit: 'SECONDS')
            }
        } catch (Exception e) {
            echo "헬스체크 중 예외 발생: ${e.message}"
            if (i < maxRetries - 1) {
                echo "재시도 중... (${i+1}/${maxRetries})"
                sleep(time: retryInterval, unit: 'SECONDS')
            }
        }
    }

    echo "API 확인 실패, 도커 로그 확인 중..."
    def logCheckResult = false

    sshagent(['ssh-ec2-ubuntu']) {
        def logStatus = sh(script: """
            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                docker logs ${server} 2>&1 | grep -i "Started"
            '
        """, returnStatus: true)

        logCheckResult = (logStatus == 0)
    }

    if (logCheckResult) {
```

```
        echo "도커 로그에서 서버 시작 확인됨: ${server}"
    } else {
        echo "도커 로그에서 서버 시작 확인 실패: ${server}"
    }

    return logCheckResult
}

pipeline {
    agent any

    options {
        timeout(time: 30, unit: 'MINUTES')
        parallelsAlwaysFailFast()
        skipDefaultCheckout(true)
    }

    // GitLab 트리거 설정
    triggers {
        gitlab(
            triggerOnPush: true,
            triggerOnMergeRequest: true,
            branchFilterType: 'NameBasedFilter',
            includeBranchesSpec: 'develop'
        )
    }

    environment {
        GIT_REPO = 'lab.ssafy.com/s12-final/S12P31B103.git'
        BRANCH_NAME = 'develop'
        PROJECT_NAME = 'S12P31B103'
        PROJECTS_DIR = '/var/jenkins_home/projects'
        DOCKER_BUILDKIT = "1"
    }

    parameters {
    choice(name: 'DEPLOYMENT_TYPE', choices: ['stable', 'canary-manual',
'canary-auto', 'promote-canary', 'rollback'], description: '배포 유형')
```

```
    string(name: 'CANARY_WEIGHT', defaultValue: '20', description: '수동 Canary 배포 시 가중치 (0-100)')
    // AUTO_FINAL_WEIGHT와 AUTO_INCREMENT_STEP은 이제 하드코딩되므로 제거하거나 주석 처리할 수 있습니다
    // string(name: 'AUTO_FINAL_WEIGHT', defaultValue: '50', description: '자동 Canary 배포 시 최종 가중치 (0-100)')
    // string(name: 'AUTO_INCREMENT_STEP', defaultValue: '10', description: '자동 배포 시 증가 단계 (0-100)')
    string(name: 'AUTO_WAIT_MINUTES', defaultValue: '2', description: '자동 배포 시 각 단계 대기 시간(분)')
    }
    // parameters {
    //     choice(name: 'DEPLOYMENT_TYPE', choices: ['stable', 'canary-manual', 'canary-auto', 'promote-canary', 'rollback'], description: '배포 유형')
    //     string(name: 'CANARY_WEIGHT', defaultValue: '20', description: '수동 Canary 배포 시 가중치 (0-100)')
    //     string(name: 'AUTO_FINAL_WEIGHT', defaultValue: '50', description: '자동 Canary 배포 시 최종 가중치 (0-100)')
    //     string(name: 'AUTO_INCREMENT_STEP', defaultValue: '10', description: '자동 배포 시 증가 단계 (0-100)')
    //     string(name: 'AUTO_WAIT_MINUTES', defaultValue: '2', description: '자동 배포 시 각 단계 대기 시간(분)')
    // }

    stages {
        // GitLab 트리거로 실행된 경우 배포 유형 결정
     stage('Determine Deployment Type') {
        steps {
            script {
                echo "GitLab 정보 확인: gitlabSourceBranch=${env.gitlabSourceBranch}, gitlabTargetBranch=${env.gitlabTargetBranch}, gitlabMergeRequestId=${env.gitlabMergeRequestId}, gitlabActionType=${env.gitlabActionType}"

                // push 또는 merge 이벤트 캐치
                def isDevPush = (env.gitlabSourceBranch == 'develop' && env.gitlabActionType == 'PUSH')
                def isDevMerge = (env.gitlabTargetBranch == 'develop' &&
```

```
                env.gitlabActionType == 'MERGE' &&
                env.gitlabMergeRequestState == 'merged')

            // 디버깅 정보 추가
            echo "isDevPush=${isDevPush}, isDevMerge=${isDevMerge}"

            // GitLab 트리거로 실행된 경우 (push 또는 merge)
            if (isDevPush || isDevMerge) {
                echo "GitLab 트리거: develop 브랜치 ${isDevPush ? '푸시' : '머
지'} → Canary 자동 배포 실행"
                env.DEPLOYMENT_TYPE = 'canary-auto'
            } else {
                echo "수동 빌드 또는 GitLab 트리거 조건 불일치: branch=${env.gitl
abSourceBranch}, targetBranch=${env.gitlabTargetBranch}, action=${env.
gitlabActionType}"
                echo "배포 유형: ${params.DEPLOYMENT_TYPE} 유형으로 배포합
니다."
                env.DEPLOYMENT_TYPE = params.DEPLOYMENT_TYPE
            }

            echo "최종 결정된 배포 유형: ${env.DEPLOYMENT_TYPE}"

            // 디버깅을 위한 전체 환경 변수 출력
            echo "모든 GitLab 관련 환경 변수:"
            sh 'env | grep -i gitlab || echo "GitLab 환경 변수 없음"'
        }
    }
}

    stage('Checkout') {
        steps {
            script {
                sh "mkdir -p ${PROJECTS_DIR}"

                def projectExists = fileExists "${PROJECTS_DIR}/${PROJECT_
NAME}"
                def isGitRepo = false
```

```
                if (projectExists) {
                    isGitRepo = sh(script: "cd ${PROJECTS_DIR}/${PROJECT_N
AME} && git rev-parse --is-inside-work-tree 2>/dev/null", returnStatus: tru
e) == 0
                }

                if (projectExists && isGitRepo) {
                    dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
                        withCredentials([usernamePassword(credentialsId: 'gitlab',
usernameVariable: 'GIT_USER', passwordVariable: 'GIT_PASS')]) {
                            sh "git remote set-url origin https://${GIT_USER}:${GIT_
PASS}@lab.ssafy.com/s12-final/S12P31B103.git"
                            sh "git fetch --depth=1 origin ${BRANCH_NAME}"
                            sh "git reset --hard origin/${BRANCH_NAME}"
                        }
                    }
                } else {
                    // 폴더가 없거나 Git 저장소가 아닌 경우 새로 클론
                    if (projectExists) {
                        sh "rm -rf ${PROJECTS_DIR}/${PROJECT_NAME}"  // 기존
폴더 삭제
                    }
                    dir("${PROJECTS_DIR}") {
                        withCredentials([usernamePassword(credentialsId: 'gitlab',
usernameVariable: 'GIT_USER', passwordVariable: 'GIT_PASS')]) {
                            sh "git clone --depth=1 -b ${BRANCH_NAME} https://
${GIT_USER}:${GIT_PASS}@lab.ssafy.com/s12-final/S12P31B103.git ${PRO
JECT_NAME}"
                        }
                    }
                }

                // 원격 서버 코드 업데이트 - 항상 실행되어야 함
                sshagent(['ssh-ec2-ubuntu']) {
                    withCredentials([usernamePassword(credentialsId: 'gitlab', u
sernameVariable: 'GIT_USER', passwordVariable: 'GIT_PASS')]) {
                        // 원격 서버의 로컬 변경사항 확인 및 처리 후 pull 수행
                        def updateResult = sh(script: """
```

```
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                        cd /home/ubuntu/project/S12P31B103 &&
                        echo "로컬 변경사항 확인 및 처리 시작" &&

                        # 현재 로컬 변경사항 확인
                        if [[ \$(git status --porcelain | wc -l) -gt 0 ]]; then
                            echo "로컬 변경사항이 있습니다. 백업 후 스태시합니다." &&

                            # 변경된 파일들 백업 디렉토리 생성
                            mkdir -p .git/backup/\$(date +%Y%m%d_%H%M%S) &&

                            # 변경된 파일들 백업
                            git status --porcelain | awk "{\$1=\\"\\"; print \$0}" | while read file; do
                                if [[ -f "\${file## }" ]]; then
                                    cp "\${file## }" ".git/backup/\$(date +%Y%m%d_%H%M%S)/\${file## }"
                                    echo "\${file## } 백업 완료"
                                fi
                            done &&

                            # 변경사항 스태시
                            git stash save "자동 백업: 파이프라인 실행 시 변경사항 - \$(date +%Y%m%d_%H%M%S)" &&
                            echo "변경사항을 스태시했습니다."
                        else
                            echo "로컬 변경사항이 없습니다."
                        fi &&

                        # 원격 저장소 설정 및 pull
                        git remote set-url origin https://${GIT_USER}:${GIT_PASS}@lab.ssafy.com/s12-final/S12P31B103.git &&
                        git pull origin ${BRANCH_NAME} &&

                        echo "최신 코드 업데이트 완료"
```

```
                    '
                """, returnStatus: true)

                if (updateResult != 0) {
                    error "원격 서버 코드 업데이트 실패. 파이프라인을 중단합니
다."
                } else {
                    echo "원격 서버 코드 업데이트 성공"
                }
            }
        }
    }
}

    stage('Check Nginx Config') {
        steps {
            script {
                def nginxConfExists = fileExists "${PROJECTS_DIR}/${PROJEC
T_NAME}/nginx/conf.d/default.conf"

                if (!nginxConfExists) {
                    echo "Nginx Canary 설정 파일이 없습니다. 새로 생성합니다."
                    sh "mkdir -p ${PROJECTS_DIR}/${PROJECT_NAME}/nginx/c
onf.d"

                    // 기본 Canary 설정 파일 생성
                    writeFile file: "${PROJECTS_DIR}/${PROJECT_NAME}/nginx/
conf.d/default.conf", text: '''
# Canary 배포를 위한 백엔드 서비스 구성 (Spring)
upstream canary_server {
    server server-stable:8080 weight=9;  # 90% 트래픽
    server server-canary:8080 weight=1;  # 10% 트래픽
}

# Canary 배포를 위한 프론트엔드 서비스 구성 (Next.js)
upstream canary_client {
    server client-stable:3000 weight=8;  # 80% 트래픽
```

```
    server client-canary:3000 weight=2;  # 20% 트래픽
}

# HTTPS 설정
server {
    listen 443 ssl;
    server_name k12b103.p.ssafy.io;

    # SSL 인증서 설정
    ssl_certificate /etc/letsencrypt/live/k12b103.p.ssafy.io/fullchain.pem;
    ssl_certificate_key /etc/letsencrypt/live/k12b103.p.ssafy.io/privkey.pem;

    # SSL 보안 설정
    ssl_protocols TLSv1.2 TLSv1.3;
    ssl_prefer_server_ciphers off;
    ssl_ciphers 'HIGH:!aNULL:!MD5';

    # API 요청 처리
    location /api {
        proxy_pass http://canary_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection 'upgrade';
        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;
        # 타임아웃 설정
        proxy_connect_timeout 300;
        proxy_send_timeout 300;
        proxy_read_timeout 300;
    }

    # 중복된 /api/api 경로 처리
    location /api/api/ {
        # 첫 번째 /api/를 제거
        rewrite ^/api/api/(.*)$ /api/$1 break;
        proxy_pass http://canary_server;
```

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

    # 웹소켓 라우팅 설정
    location /ws {
        proxy_pass http://canary_server;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
        proxy_set_header Connection "upgrade";
        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_read_timeout 3600s;
        proxy_send_timeout 3600s;
        proxy_buffering off;
    }

    # Next.js의 정적 파일 캐싱
    location /_next/static/ {
        proxy_pass http://canary_client/_next/static/;
        proxy_set_header Host $host;
        expires 30d;
        add_header Cache-Control "public, max-age=2592000";
    }

    # 프론트엔드 라우팅
    location / {
        proxy_pass http://canary_client;
        proxy_http_version 1.1;
        proxy_set_header Upgrade $http_upgrade;
```

```
        proxy_set_header Connection 'upgrade';
        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;
    }
}

# HTTP → HTTPS 리다이렉션
server {
    listen 80;
    server_name k12b103.p.ssafy.io;
    return 301 https://$host$request_uri;
}
'''

                    // 원격 서버에도 설정 파일 복사
                    sshagent(['ssh-ec2-ubuntu']) {
                        sh """
                            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ss
afy.io '
                                mkdir -p /home/ubuntu/project/S12P31B103/nginx/co
nf.d
                            '

                            scp -o StrictHostKeyChecking=no ${PROJECTS_DIR}/
${PROJECT_NAME}/nginx/conf.d/default.conf ubuntu@k12b103.p.ssafy.io:/
home/ubuntu/project/S12P31B103/nginx/conf.d/
                        """
                    }
                }

                echo "Nginx Canary 설정 파일이 준비되었습니다."
            }
        }
    }

    stage('Prepare Environment Variables') {
```

```
        steps {
            script {
                dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
                    withCredentials([
                        string(credentialsId: 'mysql-root-password', variable: 'MY
SQL_ROOT_PASSWORD'),
                        string(credentialsId: 'mysql-database', variable: 'MYSQL_D
ATABASE'),
                        string(credentialsId: 'mysql-app-user', variable: 'MYSQL_U
SER'),
                        string(credentialsId: 'mysql-app-password', variable: 'MYS
QL_PASSWORD'),
                        string(credentialsId: 'my-db', variable: 'MY_DB'),
                        string(credentialsId: 'jwt-secret', variable: 'JWT_SECRET'),
                        string(credentialsId: 'jwt-access-token-validity', variable:
'JWT_ACCESS_TOKEN_VALIDITY'),
                        string(credentialsId: 'jwt-refresh-token-validity', variable:
'JWT_REFRESH_TOKEN_VALIDITY'),
                        string(credentialsId: 'spring-data-redis-host', variable: 'SP
RING_DATA_REDIS_HOST'),
                        string(credentialsId: 'spring-data-redis-port', variable: 'SP
RING_DATA_REDIS_PORT'),
                        string(credentialsId: 'spring-data-redis-password', variabl
e: 'SPRING_DATA_REDIS_PASSWORD'),
                        string(credentialsId: 'next-public-api-url', variable: 'NEXT_
PUBLIC_API_URL'),
                        string(credentialsId: 'aws-access-key', variable: 'CLOUD_
AWS_CREDENTIALS_ACCESS_KEY'),
                        string(credentialsId: 'aws-secret-key', variable: 'CLOUD_A
WS_CREDENTIALS_SECRET_KEY'),
                        string(credentialsId: 'aws-s3-bucket', variable: 'CLOUD_A
WS_S3_BUCKET'),
                        string(credentialsId: 'kakao-client-id', variable: 'KAKAO_CL
IENT_ID'),
                        string(credentialsId: 'kakao-client-secret', variable: 'KAKA
O_CLIENT_SECRET'),
                        string(credentialsId: 'kakao-redirect-uri', variable: 'KAKAO
_REDIRECT_URI'),
```

```
                    string(credentialsId: 'claude-api-key', variable: 'CLAUDE_A
PIKEY')
                ]) {
                    // Docker Compose용 .env 파일 생성
                    sh """
                        echo "MYSQL_ROOT_PASSWORD=${MYSQL_ROOT_PA
SSWORD}" > .env
                        echo "MYSQL_DATABASE=${MYSQL_DATABASE}" >> .
env
                        echo "MYSQL_USER=${MYSQL_USER}" >> .env
                        echo "MYSQL_PASSWORD=${MYSQL_PASSWORD}" >
> .env
                        echo "MY_DB=${MY_DB}" >> .env
                        echo "JWT_SECRET=${JWT_SECRET}" >> .env
                        echo "JWT_ACCESS_TOKEN_VALIDITY=${JWT_ACCESS
_TOKEN_VALIDITY}" >> .env
                        echo "JWT_REFRESH_TOKEN_VALIDITY=${JWT_REFRES
H_TOKEN_VALIDITY}" >> .env
                        echo "SPRING_DATA_REDIS_HOST=${SPRING_DATA_RE
DIS_HOST}" >> .env
                        echo "SPRING_DATA_REDIS_PORT=${SPRING_DATA_RE
DIS_PORT}" >> .env
                        echo "SPRING_DATA_REDIS_PASSWORD=${SPRING_DA
TA_REDIS_PASSWORD}" >> .env

                        echo "CLOUD_AWS_CREDENTIALS_ACCESS_KEY=${CL
OUD_AWS_CREDENTIALS_ACCESS_KEY}" >> .env
                        echo "CLOUD_AWS_CREDENTIALS_SECRET_KEY=${CL
OUD_AWS_CREDENTIALS_SECRET_KEY}" >> .env
                        echo "CLOUD_AWS_S3_BUCKET=${CLOUD_AWS_S3_B
UCKET}" >> .env
                        echo "CLOUD_AWS_REGION_STATIC=ap-northeast-2" >
> .env
                        echo "CLOUD_AWS_STACK_AUTO=false" >> .env

                        echo "KAKAO_CLIENT_ID=${KAKAO_CLIENT_ID}" >> .e
nv
                        echo "KAKAO_CLIENT_SECRET=${KAKAO_CLIENT_SEC
```

```
RET}" >> .env
                    echo "KAKAO_REDIRECT_URI=${KAKAO_REDIRECT_UR
I}" >> .env

                    echo "CLAUDE_APIKEY=${CLAUDE_APIKEY}" >> .env

                    # 추가 환경 변수 설정
                    echo "SPRING_PROFILES=development" >> .env
                    echo "SPRING_DATASOURCE_URL=jdbc:mysql://\${MY_
DB}:3306/\${MYSQL_DATABASE}" >> .env
                    echo "SPRING_DATASOURCE_USERNAME=\${MYSQL_U
SER}" >> .env
                    echo "SPRING_DATASOURCE_PASSWORD=\${MYSQL_
PASSWORD}" >> .env
                """

                // 프론트엔드용 .env 파일 생성
                sh "mkdir -p client"
                sh "echo \"NEXT_PUBLIC_API_URL=${NEXT_PUBLIC_API_
URL}\" > client/.env"

                // 원격 서버에 환경 변수 파일 복사
                echo "원격 서버에 환경 변수 파일 복사 시작"
                sshagent(['ssh-ec2-ubuntu']) {
                    def scpResult = sh(script: """
                        scp -o StrictHostKeyChecking=no .env ubuntu@k12b1
03.p.ssafy.io:/home/ubuntu/project/S12P31B103/ || echo "SCP 실패"
                        scp -o StrictHostKeyChecking=no client/.env ubuntu
@k12b103.p.ssafy.io:/home/ubuntu/project/S12P31B103/client/ || echo "SC
P 실패"
                    """, returnStatus: true)

                    if (scpResult != 0) {
                        echo "SCP 명령 실행 중 오류 발생: 리턴 코드 ${scpResul
t}"
                    } else {
                        echo "환경 변수 파일 복사 성공"
                    }
```

```
                    }
                }
            }
        }
    }

    stage('Check SSH Connection') {
        steps {
            script {
                echo "SSH 연결 테스트 중..."

                sshagent(['ssh-ec2-ubuntu']) {
                    def sshCheck = sh(script: """
                        ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssaf
y.io 'echo "SSH 연결 성공"'
                    """, returnStatus: true)

                    if (sshCheck != 0) {
                        error "SSH 연결 테스트 실패: 리턴 코드 ${sshCheck}"
                    } else {
                        echo "SSH 연결 테스트 성공"
                    }
                }
            }
        }
    }

    stage('Deploy Stable') {
        when {
            expression { return env.DEPLOYMENT_TYPE == 'stable' }
        }
        steps {
            script {
                dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
                    echo "안정 버전(Stable) 배포 시작..."

                    try {
```

```
                    echo "SSH 연결 시도 중..."

                    // SSH를 통해 원격 서버에서 배포
                    sshagent(['ssh-ec2-ubuntu']) {
                        echo "SSH 연결 성공, 원격 명령 실행 준비 완료"

                        // Docker 버전 확인
                        sh """
                            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.
p.ssafy.io '
                                echo "Docker 버전 확인:"
                                docker --version
                                echo "Docker Compose 버전 확인:"
                                docker compose version
                            '
                        """

                        // 원격 서버의 프로젝트 디렉토리에서 Docker Compose 실행
                        sh """
                            echo "원격 서버에 명령 전송 중..."
                            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.
p.ssafy.io '
                                echo "원격 서버에서 명령 수신 완료"
                                cd /home/ubuntu/project/S12P31B103 &&
                                echo "디렉토리 이동 완료: \$(pwd)"
                                docker compose -f docker-compose.yml up -d --b
uild
                            '
                        """

                        echo "원격 Docker 명령 실행 완료"
                    }

                    // 가중치 설정 (100% 안정 버전으로)
                    echo "Nginx 가중치 설정 시작"
                    updateNginxWeights(100, 0)
                    echo "Nginx 가중치 설정 완료"
```

```
                } catch (Exception e) {
                    echo "배포 중 오류 발생: ${e.message}"
                    echo "스택 트레이스: ${e.getStackTrace().join('\n')}"
                    throw e
                }
            }
        }
    }

    stage('Deploy Canary') {
        when {
            expression { return env.DEPLOYMENT_TYPE == 'canary-manual' ||
| env.DEPLOYMENT_TYPE == 'canary-auto' }
        }
        steps {
            script {
                dir("${PROJECTS_DIR}/${PROJECT_NAME}") {
                    echo "Canary 버전 배포 시작..."

                    // SSH를 통해 원격 서버에서 배포
                    sshagent(['ssh-ec2-ubuntu']) {
                        // 안정 버전이 실행 중인지 확인
                        def stableRunning = sh(script: """
                            ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ss
afy.io '
                                docker ps | grep server-stable || echo "not running"
                            '
                        """, returnStdout: true).trim() != "not running"

                        if (!stableRunning) {
                            echo "안정 버전이 실행 중이지 않습니다. 먼저 안정 버전을 배
포합니다."
                            sh """
                                ssh -o StrictHostKeyChecking=no ubuntu@k12b103.
p.ssafy.io '
                                    cd /home/ubuntu/project/S12P31B103 &&
                                    docker compose -f docker-compose.yml up -d
```

```
                        '
                    """
                }

                // Canary 배포
                sh """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.io '
                        cd /home/ubuntu/project/S12P31B103 &&
                        docker compose -f docker-compose.canary.yml up -d --build
                    '
                """
            }

            // 스프링 서버 헬스체크
            echo "Canary 서버 헬스체크 시작..."
            if (!checkServerHealth('server-canary', '8082')) {
                error "Canary 서버 시작 실패"
            }

            // Python 서버 헬스체크 추가
            echo "Canary Python 서버 헬스체크 시작..."
            if (!checkServerHealth('python-canary', '8020')) {
                error "Canary Python 서버 시작 실패"
            }

            echo "Canary 서버들이 정상적으로 시작되었습니다."
        }
    }
}

stage('Manual Traffic Shift') {
    when {
        expression { return env.DEPLOYMENT_TYPE == 'canary-manual' }
    }
```

```
        steps {
            script {
                def canaryWeight = params.CANARY_WEIGHT.toInteger()
                def stableWeight = 100 - canaryWeight

                echo "수동 트래픽 조정: Stable=${stableWeight}%, Canary=${ca
naryWeight}%"
                updateNginxWeights(stableWeight, canaryWeight)
            }
        }
    }
    stage('Auto Progressive Traffic Shift') {
        when {
            expression { return env.DEPLOYMENT_TYPE == 'canary-auto' }
        }
        steps {
            script {
                // 트래픽 단계 설정 (10%, 40%, 70%)
                def trafficSteps = [10, 40, 70]

                // 대기 시간을 초 단위로 변경
                def waitTimeSeconds = 30
                env.ACTUAL_WAIT_TIME = "30초" // 전역 변수로 저장

                echo "자동 점진적 Canary 배포 시작 - 트래픽 단계: ${trafficSteps.jo
in('% → ')}%, 대기 시간: ${waitTimeSeconds}초"

                // 초기 가중치 설정 (첫 단계: 10%)
                def stableWeight = 100 - trafficSteps[0]
                def canaryWeight = trafficSteps[0]

                updateNginxWeights(stableWeight, canaryWeight)
                echo "초기 트래픽 비율 설정: Stable=${stableWeight}%, Canary=
${canaryWeight}%"

                // 모니터링 대기
                echo "${waitTimeSeconds}초 동안 모니터링 중..."
                sleep time: waitTimeSeconds, unit: 'SECONDS'
```

```
                // 초기 건전성 확인
                if (!monitorCanaryDeployment(canaryWeight)) {
                    error "초기 Canary 배포에서 문제가 발견되었습니다. 롤백을 시작합
니다."
                }

                // 점진적으로 트래픽 증가 (40%, 70%)
                for (int i = 1; i < trafficSteps.size(); i++) {
                    def weight = trafficSteps[i]
                    stableWeight = 100 - weight
                    canaryWeight = weight

                    echo "Canary 트래픽 비율을 ${weight}%로 증가시킵니다..."
                    updateNginxWeights(stableWeight, canaryWeight)

                    // 모니터링 대기
                    echo "${waitTimeSeconds}초 동안 모니터링 중..."
                    sleep time: waitTimeSeconds, unit: 'SECONDS'

                    // 각 단계마다 건전성 확인
                    if (!monitorCanaryDeployment(weight)) {
                        echo "Canary 배포 중 문제가 발견되었습니다. 롤백을 시작합니
다."
                        updateNginxWeights(100, 0)
                        error "Canary 배포 ${weight}% 단계에서 실패하여 롤백했습니
다."
                    }
                }
                def finalCanaryPercent = trafficSteps.last()
                env.FINAL_CANARY_PERCENT = finalCanaryPercent.toString()
                echo "최종 Canary 트래픽 비율 ${env.FINAL_CANARY_PERCEN
T}%를 환경 변수에 저장했습니다."
                echo "Canary 배포가 성공적으로 70%까지 증가했습니다! 100%로 승
격하려면 'promote-canary' 배포 유형으로 수동 빌드를 실행하세요."
            }
        }
    }
```

```
        stage('Promote Canary to Stable') {
            when {
                expression { return env.DEPLOYMENT_TYPE == 'promote-canary'
}
            }
            steps {
                script {
                    echo "Canary 버전을 정식 버전으로 승격합니다 (100% 트래픽 전환)."

                    // 100% 트래픽을 Canary로 전환
                    updateNginxWeights(0, 100)

                    // 여기에 Canary 이미지를 Stable로 승격시키는 로직을 추가할 수 있음
                    // 예: 이미지 태그 변경, 설정 복사 등
                }
            }
        }

        stage('Rollback') {
            when {
                expression { return env.DEPLOYMENT_TYPE == 'rollback' }
            }
            steps {
                script {
                    echo "안정 버전으로 롤백합니다 (100% 트래픽을 Stable로 전환)."

                    // 100% 트래픽을 Stable로 전환
                    updateNginxWeights(100, 0)
                }
            }
        }

        stage('Verify Deployment') {
            steps {
                script {
                    echo "배포 검증 중..."
```

```
                // 배포 유형에 따른 예상 가중치 설정
                def expectedStableServer = 0
                def expectedCanaryServer = 0
                def expectedStableClient = 0
                def expectedCanaryClient = 0
                def expectedStablePython = 0
                def expectedCanaryPython = 0

                if (env.DEPLOYMENT_TYPE == 'stable' || env.DEPLOYMENT_TY
PE == 'rollback') {
                    expectedStableServer = 999
                    expectedCanaryServer = 1
                    expectedStableClient = 999
                    expectedCanaryClient = 1
                    expectedStablePython = 999
                    expectedCanaryPython = 1
                }
                else if (env.DEPLOYMENT_TYPE == 'canary-manual') {
                    def canaryPercent = params.CANARY_WEIGHT.toInteger()
                    expectedCanaryServer = (canaryPercent / 10) as int
                    expectedStableServer = ((100 - canaryPercent) / 10) as int
                    expectedCanaryClient = expectedCanaryServer
                    expectedStableClient = expectedStableServer
                    expectedCanaryPython = expectedCanaryServer
                    expectedStablePython = expectedStableServer
                    expectedCanaryPython = expectedCanaryServer
                    expectedStablePython = expectedStableServer
                }
                else if (env.DEPLOYMENT_TYPE == 'canary-auto') {
                    // 저장된 마지막 트래픽 값 사용
                    def canaryPercent = env.FINAL_CANARY_PERCENT ? env.FI
NAL_CANARY_PERCENT.toInteger() : 70
                    expectedCanaryServer = (canaryPercent / 10) as int
                    expectedStableServer = ((100 - canaryPercent) / 10) as int
                    expectedCanaryClient = expectedCanaryServer
                    expectedStableClient = expectedStableServer
                    expectedCanaryPython = expectedCanaryServer
```

```
                expectedStablePython = expectedStableServer
            }
            else if (env.DEPLOYMENT_TYPE == 'promote-canary') {
                expectedStableServer = 1
                expectedCanaryServer = 999
                expectedStableClient = 1
                expectedCanaryClient = 999
                expectedStablePython = 1
                expectedCanaryPython = 999
            }

            echo "예상 가중치: stableServer=${expectedStableServer}, canaryServer=${expectedCanaryServer}, stableClient=${expectedStableClient}, canaryClient=${expectedCanaryClient}, stablePython=${expectedStablePython}, canaryPython=${expectedCanaryPython}"

            // 가중치 확인 전에 잠시 대기 (Nginx 설정이 적용될 시간 확보)
            sleep(time: 5, unit: 'SECONDS')

            // 현재 설정된 가중치 확인
            def weights = getCurrentWeights()
            echo "현재 트래픽 가중치: ${weights}"

            // 가중치 확인 및 검증
            def isServerWeightCorrect = (Math.abs(weights.stableServer - expectedStableServer) <= 1 &&
                                         Math.abs(weights.canaryServer - expectedCanaryServer) <= 1)

            def isClientWeightCorrect = (Math.abs(weights.stableClient - expectedStableClient) <= 1 &&
                                         Math.abs(weights.canaryClient - expectedCanaryClient) <= 1)

            def isPythonWeightCorrect = (Math.abs(weights.stablePython - expectedStablePython) <= 1 &&
                                         Math.abs(weights.canaryPython - expectedCanaryPython) <= 1)
```

```
                if (!isServerWeightCorrect || !isClientWeightCorrect || !isPython
WeightCorrect) {
                    echo "가중치가 일치하지 않습니다. 다시 설정을 시도합니다."

                    if (env.DEPLOYMENT_TYPE == 'stable' || env.DEPLOYMENT_
TYPE == 'rollback') {
                        updateNginxWeights(100, 0)
                    } else if (env.DEPLOYMENT_TYPE == 'promote-canary') {
                        updateNginxWeights(0, 100)
                    } else if (env.DEPLOYMENT_TYPE == 'canary-manual') {
                        def canaryPercent = params.CANARY_WEIGHT.toInteger()
                        updateNginxWeights(100 - canaryPercent, canaryPercent)
                    } else if (env.DEPLOYMENT_TYPE == 'canary-auto') {
                        def canaryPercent = params.AUTO_FINAL_WEIGHT.toInte
ger()
                        updateNginxWeights(100 - canaryPercent, canaryPercent)
                    }

                    // 다시 대기 (설정이 적용될 시간을 더 길게 제공)
                    sleep(time: 10, unit: 'SECONDS')

                    // 재설정 후 가중치 확인
                    weights = getCurrentWeights()
                    echo "재설정 후 트래픽 가중치: ${weights}"

                    // 최종 검증
                    isServerWeightCorrect = (Math.abs(weights.stableServer - e
xpectedStableServer) <= 2 &&
                                             Math.abs(weights.canaryServer - expectedCa
naryServer) <= 2)

                    isClientWeightCorrect = (Math.abs(weights.stableClient - exp
ectedStableClient) <= 2 &&
                                             Math.abs(weights.canaryClient - expectedCan
aryClient) <= 2)

                    isPythonWeightCorrect = (Math.abs(weights.stablePython -
```

```
expectedStablePython) <= 2 &&
                                    Math.abs(weights.canaryPython - expectedCa
naryPython) <= 2)

                    if (!isServerWeightCorrect || !isClientWeightCorrect || !isPyth
onWeightCorrect) {
                        error "트래픽 가중치가 재시도 후에도 예상과 다릅니다. 예상: Sta
ble Server=${expectedStableServer}, Canary Server=${expectedCanarySe
rver}, Stable Client=${expectedStableClient}, Canary Client=${expectedCa
naryClient}, Stable Python=${expectedStablePython}, Canary Python=${ex
pectedCanaryPython}"
                    }
                }

                echo "트래픽 가중치가 정상적으로 설정되었습니다!"
            }
        }
    }
}

post {
    success {
        script {
            def kTime = getKoreanTime()
            def deploymentType = env.DEPLOYMENT_TYPE
            def messagePrefix = "✅ *Canary 배포 성공*"
            def messageDetails = ""

            if (deploymentType == 'stable') {
                messagePrefix = "✅ *안정 버전 배포 성공*"
                messageDetails = "트래픽 전환 비율: Stable 100%, Canary 0%"
            } else if (deploymentType == 'canary-manual') {
                messagePrefix = "✅ *수동 Canary 배포 성공 (${params.CANAR
Y_WEIGHT}%)*"
                messageDetails = "트래픽 전환 비율: Stable ${100 - params.CAN
ARY_WEIGHT.toInteger()}%, Canary ${params.CANARY_WEIGHT}%"
            } else if (deploymentType == 'canary-auto') {
                def finalPercent = env.FINAL_CANARY_PERCENT ? env.FINAL_
```

```
CANARY_PERCENT : "70"
                messagePrefix = "✅ *자동 Canary 배포 성공 (${finalPercent}%)
*"
                messageDetails = "증가 단계: 10%, 40%, 70%, 대기 시간: 30초, 트
래픽 전환 비율: Stable ${100 - finalPercent.toInteger()}%, Canary ${finalPerc
ent}%"
            } else if (deploymentType == 'promote-canary') {
                messagePrefix = "✅ *Canary 버전 승격 성공 (100%)*"
                messageDetails = "트래픽 전환 비율: Canary 99.9%, Stable 0.1%"
            } else if (deploymentType == 'rollback') {
                messagePrefix = "✅ *Rollback 성공*"
                messageDetails = "트래픽 전환 비율: Stable 100%, Canary 0%, 롤
백 완료 시간: ${kTime}"

            }

            echo """
            ✅ ==========================================
=====
                🎉 ${deploymentType} 배포 성공! 🎉
                📋 작업명: ${env.JOB_NAME}
                🔢 빌드 번호: #${env.BUILD_NUMBER}
                🕒 완료 시간: ${kTime}
                🎯 배포 타입: ${deploymentType}
                📌 세부 정보: ${messageDetails}
            ==========================================
== ✅
            """

            // Mattermost 알림
            sshagent(['ssh-ec2-ubuntu']) {
                def Author_ID = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%an
                    '
                """, returnStdout: true).trim()
```

```
                def Author_Email = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%ae
                    '
                """, returnStdout: true).trim()

                def Commit_Hash = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%h
                    '
                """, returnStdout: true).trim()

                def Commit_Msg = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%s
                    '
                """, returnStdout: true).trim()

                mattermostSend(
                    color: 'good',
                    message: """${messagePrefix}
    - *작업명*: ${env.JOB_NAME}
    - *빌드*: #${env.BUILD_NUMBER}
    - *브랜치*: ${BRANCH_NAME}
    - *커밋*: ${Commit_Hash} - ${Commit_Msg}
    - *작성자*: ${Author_ID} <${Author_Email}>
    - *완료 시간*: ${kTime}
    - *배포 타입*: ${deploymentType}
    - *세부 정보*: ${messageDetails}""",
                    endpoint: 'https://meeting.ssafy.com/hooks/tj3cixzzc7dobce
fc3pknxjsjh',
```

```
                channel: 'B103-Build-Noti'
            )
        }
    }
}

failure {
    script {
        def kTime = getKoreanTime()
        def deploymentType = env.DEPLOYMENT_TYPE

        echo """
        ❌ ==========================================
=====
          ⚠️ ${deploymentType} 배포 실패! ⚠️
          📋 작업명: ${env.JOB_NAME}
          🔢 빌드 번호: #${env.BUILD_NUMBER}
          🕑 실패 시간: ${kTime}
        ==========================================
== ❌
        """

        // 실패 시 자동 롤백 (canary 배포의 경우)
        if (deploymentType.startsWith('canary') || deploymentType == 'pr
omote-canary') {
            echo "배포 실패로 인한 자동 롤백 시도 중..."
            updateNginxWeights(100, 0)
        }

        // Mattermost 알림
        sshagent(['ssh-ec2-ubuntu']) {
            def Author_ID = sh(script: """
                ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                    cd /home/ubuntu/project/S12P31B103 &&
                    git show -s --pretty=%an
                '
            """, returnStdout: true).trim()
```

```
                def Author_Email = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%ae
                    '
                """, returnStdout: true).trim()

                def Commit_Hash = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%h
                    '
                """, returnStdout: true).trim()

                def Commit_Msg = sh(script: """
                    ssh -o StrictHostKeyChecking=no ubuntu@k12b103.p.ssafy.i
o '
                        cd /home/ubuntu/project/S12P31B103 &&
                        git show -s --pretty=%s
                    '
                """, returnStdout: true).trim()

                mattermostSend(
                    color: 'danger',
                    message: """❌ *${deploymentType} 배포 실패*
    - *작업명*: ${env.JOB_NAME}
    - *빌드*: #${env.BUILD_NUMBER}
    - *브랜치*: ${BRANCH_NAME}
    - *커밋*: ${Commit_Hash} - ${Commit_Msg}
    - *작성자*: ${Author_ID} <${Author_Email}>
    - *실패 시간*: ${kTime}
    - *로그 확인*: ${env.BUILD_URL}console""",
                    endpoint: 'https://meeting.ssafy.com/hooks/tj3cixzzc7dobce
fc3pknxjsjh',
                    channel: 'B103-Build-Noti'
```

```
                )
            }
        }
    }

    always {
        echo """
        🧹 ============================================
===
          작업 공간 정리 중...
          🗑️ 임시 파일 및 아티팩트 제거
        ============================================
= 🧹
        """
        cleanWs()
    }
  }
}
```