# 포팅 메뉴얼

## 1. 개발 및 배포 환경 버전

| 항목 | 종류 / 제품 / 버전 |
|---|---|
| JVM | OpenJDK 17.0.12 |
| IDE | IntelliJ IDEA 2023.3.8 |
| WAS | Spring Boot 3.5.4 |
| Node.js | v22.18.0 |
| npm | 10.9.3 |
| pnpm | 10.14.0 |
| TypeScript | 5.8.3 |
| Redis | 8.0.3 |
| MySQL | 9.4.0 |
| Ubuntu | 22.04.4 LTS |
| Docker | 28.4.0 |
| Nginx | 1.18.0 |

## 2. 빌드 시 사용되는 환경변수

**application-dev.yml**

```
server:
  port: 8080

spring:
  application:
    name: majoong-service
  datasource:
    url: jdbc:mysql://j13e105.p.ssafy.io:3306/machimnae_db
    username: [DB_USERNAME]
    password: [DB_PASSWORD]
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
```

```
  hibernate:
    ddl-auto: update
  show-sql: true
  properties:
    hibernate:
      dialect: org.hibernate.dialect.MySQL8Dialect
      format_sql: true
batch:
  jdbc:
    initialize-schema: never
    table-prefix: BATCH_
  job:
    enabled: false

security:
  oauth2:
    client:
      registration:
        kakao:
          client-id: [KAKAO_CLIENT_ID]
          client-secret: [KAKAO_CLIENT_SECRET]
          client-authentication-method: client_secret_post
          authorization-grant-type: authorization_code
          redirect-uri: https://api-test.majoong.site/login/oauth2/code/kakao
          scope:
            - profile_nickname
            - account_email
          client-name: kakao
      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
data:
  redis:
    host: j13e105.p.ssafy.io
    port: 6379
```

```
    password: [REDIS_PASSWORD]
  servlet:
    multipart:
      max-file-size: 50MB
      max-request-size: 200MB

jwt:
  secret-key: [JWT_SECRET_KEY]
  access-expire-time: 3600000
  refresh-expire-time: 604800000

app:
  redirect-uri: https://test.majoong.site/login/callback
  frontend: https://test.majoong.site

verification:
  api:
    key: [VERIFICATION_API_KEY]

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

# ====== 블록체인(수탁형) 추가 ======
chain:
  rpcUrl: "https://sepolia.infura.io/v3/[INFURA_PROJECT_ID]"
  chainId: 11155111
  tokenAddress: [TOKEN_ADDRESS]
  factoryAddress: [FACTORY_ADDRESS]
  krwPerToken: 100

web3:
  adminPrivateKey: [WEB3_ADMIN_PRIVATE_KEY]
  keystoreDir: "./keystore"

security:
  keystoreEncryptKey: [KEYSTORE_ENCRYPT_KEY]
```

```
cloud:
  aws:
    credentials:
      access-key: [AWS_ACCESS_KEY]
      secret-key: [AWS_SECRET_KEY]
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: e105

finapi:
  base-url: https://finopenapi.ssafy.io/ssafy/api/v1
  api-key: [FINAPI_KEY]
  institution-code: "00100"
  fintech-app-no: "001"
  account-type-unique-no: [FINAPI_ACCOUNT_TYPE_NO]
  admin-account-no: [FINAPI_ADMIN_ACCOUNT_NO]
  admin-user-key: [FINAPI_ADMIN_USER_KEY]

kakao:
  MapApi: [KAKAO_MAP_API_KEY]

openai:
  api-key: [OPENAI_API_KEY]
  text-model: gpt-4o-mini
  http-client:
    read-timeout: 3000
    connect-timeout: 3000
  urls:
    base-url: https://gms.ssafy.io/gmsapi
    create-text-url: /api.openai.com/v1/chat/completions
  image-url: /gmsapi/generativelanguage.googleapis.com/v1beta/models/i
magen-3.0-generate-002:predict
  image-base-url: https://gms.ssafy.io
```

```
kakaopay:
  secretKey: [KAKAOPAY_SECRET_KEY]
  cid: TC0ONETIME
  url:
    ready: https://open-api.kakaopay.com/online/v1/payment/ready
    approveRedirect: https://open-api.kakaopay.com/online/v1/payment/app
rove
    approve: https://api-test.majoong.site/api/v1/kakao-pay/approve
    cancel: https://test.majoong.site/kakao-pay/cancel
    fail: https://test.majoong.site/kakao-pay/fail

jobs:
  scheduling:
    enabled: true
```

**application-prod.yml**

```
server:
  port: 8080

spring:
  application:
    name: majoong-service
  datasource:
    url: jdbc:mysql://j13e105.p.ssafy.io:3306/machimnae_db
    username: [DB_USERNAME]
    password: [DB_PASSWORD]
    driver-class-name: com.mysql.cj.jdbc.Driver
  jpa:
    hibernate:
      ddl-auto: update
    show-sql: true
    properties:
      hibernate:
        dialect: org.hibernate.dialect.MySQL8Dialect
        format_sql: true
  batch:
```

```
  jdbc:
    initialize-schema: never
    table-prefix: BATCH_
  job:
    enabled: false
jobs:
  scheduling:
    enabled: true

security:
  oauth2:
    client:
      registration:
        kakao:
          client-id: [KAKAO_CLIENT_ID]
          client-secret: [KAKAO_CLIENT_SECRET]
          client-authentication-method: client_secret_post
          authorization-grant-type: authorization_code
          redirect-uri: https://api.majoong.site/login/oauth2/code/kakao
          scope:
            - profile_nickname
            - account_email
          client-name: kakao
      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
data:
  redis:
    host: j13e105.p.ssafy.io
    port: 6379
    password: [REDIS_PASSWORD]

servlet:
  multipart:
    max-file-size: 50MB
```

```
      max-request-size: 200MB

jwt:
  secret-key: [JWT_SECRET_KEY]
  access-expire-time: 3600000
  refresh-expire-time: 604800000

app:
  redirect-uri: https://www.majoong.site/login/callback
  frontend: https://www.majoong.site

verification:
  api:
    key: [VERIFICATION_API_KEY]

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

# ====== 블록체인(수탁형) 추가 ======
chain:
  rpcUrl: "https://sepolia.infura.io/v3/[INFURA_PROJECT_ID]"
  chainId: 11155111
  tokenAddress: [TOKEN_ADDRESS]
  factoryAddress: [FACTORY_ADDRESS]
  krwPerToken: 100

web3:
  adminPrivateKey: [WEB3_ADMIN_PRIVATE_KEY]
  keystoreDir: "./keystore"

security:
  keystoreEncryptKey: [KEYSTORE_ENCRYPT_KEY]

cloud:
  aws:
    credentials:
```

```
      access-key: [AWS_ACCESS_KEY]
      secret-key: [AWS_SECRET_KEY]
    region:
      static: ap-northeast-2
    stack:
      auto: false
    s3:
      bucket: e105

finapi:
  base-url: https://finopenapi.ssafy.io/ssafy/api/v1
  api-key: [FINAPI_KEY]
  institution-code: "00100"
  fintech-app-no: "001"
  account-type-unique-no: [FINAPI_ACCOUNT_TYPE_NO]
  admin-account-no: [FINAPI_ADMIN_ACCOUNT_NO]
  admin-user-key: [FINAPI_ADMIN_USER_KEY]

kakao:
  MapApi: [KAKAO_MAP_API_KEY]

openai:
  api-key: [OPENAI_API_KEY]
  text-model: gpt-4o-mini
  http-client:
    read-timeout: 3000
    connect-timeout: 3000
  urls:
    base-url: https://gms.ssafy.io/gmsapi
    create-text-url: /api.openai.com/v1/chat/completions
  image-url: /gmsapi/generativelanguage.googleapis.com/v1beta/models/i
magen-3.0-generate-002:predict
  image-base-url: https://gms.ssafy.io

kakaopay:
  secretKey: [KAKAOPAY_SECRET_KEY]
  cid : TC0ONETIME
  url:
```

```
    ready: https://open-api.kakaopay.com/online/v1/payment/ready
    approveRedirect: https://open-api.kakaopay.com/online/v1/payment/app
rove
    approve: https://api.majoong.site/api/v1/kakao-pay/approve
    cancel: https://www.majoong.site/kakao-pay/cancel
    fail: https://www.majoong.site/kakao-pay/fail

jobs:
  scheduling:
    enabled: true
```

**.env.production.build**

```
NEXT_PUBLIC_API_URL=https://api-test.majoong.site
```

**.env.development.build**

```
NEXT_PUBLIC_API_URL=https://api.majoong.site
```

**.env.runtime**

```
CLOVA_OCR_INVOKE_URL=[CLOVA_OCR_INVOKE_URL]
CLOVA_OCR_SECRET=[CLOVA_OCR_SECRET]
OPENAI_API_KEY=[OPENAI_API_KEY]
OPENAI_API_URL=[OPENAI_API_URL]

HORSE_API_SERVICE_KEY=[HORSE_API_SERVICE_KEY]
```

# 3. JenkinsFile

```
pipeline {
    agent any
```

```
    environment {
        BACKEND_DIR      = 'backend'
        FRONTEND_DIR     = 'frontend'
        DEV_BACK_CONTAINER   = 'majoong-backend-dev'
        DEV_FRONT_CONTAINER  = 'majoong-frontend-dev'
        PROD_BACK_CONTAINER  = 'majoong-backend-prod'
        PROD_FRONT_CONTAINER = 'majoong-frontend-prod'
        DEV_BACK_PORT    = '8081'
        DEV_FRONT_PORT   = '3001'
        PROD_BACK_PORT   = '8082'
        PROD_FRONT_PORT  = '3000'
        TEST_NETWORK     = 'test-network'
        PROD_NETWORK     = 'prod-network'
        LOG_FILE = 'ci.log'
    }

    options {
        timestamps()
        disableConcurrentBuilds()
    }

    stages {
        stage('Init Log') {
            steps {
                echo "🗒️  Init Log: 워크스페이스 로그 파일 초기화"
                sh '''
                set -eu
                : "${WORKSPACE:?}"
                rm -f "$WORKSPACE/${LOG_FILE:-ci.log}" || true
                touch "$WORKSPACE/${LOG_FILE:-ci.log}"
                echo "[INIT] ci.log created at $(date -u +%Y-%m-%dT%H:%M:%
SZ)" >> "$WORKSPACE/${LOG_FILE:-ci.log}"
                '''
            }
        }

        stage('Detect Changes') {
```

```
        steps {
            echo "🔍 Detect Changes: 변경 파일 스캔"
            script {
                def range = env.GIT_PREVIOUS_SUCCESSFUL_COMMIT ? "${e
nv.GIT_PREVIOUS_SUCCESSFUL_COMMIT}..HEAD" : "HEAD~1..HEAD"
                def changedFiles = sh(script: "git diff --name-only ${range} || t
rue", returnStdout: true).trim()

                if (!changedFiles) {
                    echo "❎ 변경된 파일이 없습니다. 스킵합니다."
                    env.BACK_CHANGED  = 'false'
                    env.FRONT_CHANGED = 'false'
                    env.CHAIN_CHANGED  = 'false'
                } else {
                    echo "📄 변경 파일 목록:\n${changedFiles}"
                    def lines = changedFiles.split('\\n') as List<String>
                    env.BACK_CHANGED  = (lines.any { it.startsWith('backend/')
}).toString()
                    env.FRONT_CHANGED = (lines.any { it.startsWith('frontend/')
}).toString()
                    env.CHAIN_CHANGED  = (lines.any { it.startsWith('blockchai
n/') }).toString()
                }

                echo "🧭 변경 요약 → BACK_CHANGED=${env.BACK_CHANGE
D}, FRONT_CHANGED=${env.FRONT_CHANGED}, CHAIN_CHANGED=${en
v.CHAIN_CHANGED}, range=${range}."
            }
        }
    }

    stage('Detect Branch') {
        steps {
            echo "🌿 Detect Branch: 브랜치 이름 확인"
            script {
                def resolved = env.BRANCH_NAME?.trim()
                if (!resolved) {
                    resolved = env.GIT_REF?.replaceFirst(/^refs\\/heads\\//,'')?.tri
```

```
m()
                }
                if (!resolved) {
                    resolved = sh(script: "git name-rev --name-only HEAD || git r
ev-parse --abbrev-ref HEAD",
                                  returnStdout: true).trim()
                }
                env.BRANCH_NAME = resolved
                echo "▶ Active Branch = ${env.BRANCH_NAME}"
            }
        }
    }

    stage('Prepare Secret') {
        steps {
            echo "🔐 Prepare Secret: application.yml 주입"
            sh "mkdir -p ${BACKEND_DIR}/src/main/resources"
            script {
                if (env.BRANCH_NAME == 'main') {
                    echo "🚀 환경: prod (main)"
                    withCredentials([file(credentialsId: 'SECRETFILE_PROD', vari
able: 'ENV_YML')]) {
                    sh """
                        set -eu
                        cp "\$ENV_YML" "${env.BACKEND_DIR}/src/main/resourc
es/application.yml" >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                        chmod 600 "${env.BACKEND_DIR}/src/main/resources/ap
plication.yml"     >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                        echo "[SECRET] prod application.yml installed"
>> "\$WORKSPACE/${LOG_FILE}"
                    """
                    }
                } else if (env.BRANCH_NAME == 'dev') {
                    echo "🚀 환경: dev (dev)"
                    withCredentials([file(credentialsId: 'SECRETFILE_DEV', variab
le: 'ENV_YML')]) {
                    sh """
                        set -eu
```

```
                    cp "\$ENV_YML" "${env.BACKEND_DIR}/src/main/resourc
es/application.yml"  >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                    chmod 600 "${env.BACKEND_DIR}/src/main/resources/ap
plication.yml"      >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                    echo "[SECRET] dev application.yml installed"
>> "\$WORKSPACE/${LOG_FILE}"
                """
                }
              } else {
                echo "ℹ️ main/dev 외 브랜치: 시크릿 복사 생략"
              }
            }
          }
        }

        stage('Nothing to Build') {
          when { expression { env.BACK_CHANGED != 'true' && env.FRONT_C
HANGED != 'true' && env.CHAIN_CHANGED != 'true' } }
          steps {
            echo "⏭️ 변경 없음 → 모든 빌드 단계 스킵"
            script { currentBuild.result = 'NOT_BUILT' }
          }
        }

        stage('Backend Build') {
          when { expression { env.BACK_CHANGED == 'true' || env.BRANCH_
NAME == 'main' } }
          steps {
            echo "🛠️ Backend Build: Gradle 빌드 시작"
            dir("${BACKEND_DIR}") {
              script {
                try {
                  sh """#!/usr/bin/env bash
                  set -Eeuo pipefail
                  echo "[BACKEND] build start"                    >> "\$WORKS
PACE/${LOG_FILE}"
                  chmod +x ./gradlew                          >> "\$WORKSPA
CE/${LOG_FILE}" 2>&1
```

```
                    set -x
                    ./gradlew --no-daemon build -x test --stacktrace --warnin
g-mode all --info \
                    2>&1 | tee -a "\$WORKSPACE/${LOG_FILE}"
                    ec=\$?
                    set +x
                    echo "[BACKEND] build exit=\${ec}"              >> "\$WOR
KSPACE/${LOG_FILE}"
                    exit "\${ec}"
                    """
                    echo "✅ Backend Build: 성공"
                } catch (err) {
                    sh "echo '[ERROR] Backend Build failed: ${err}' >> \"$WO
RKSPACE/${LOG_FILE}\""
                    echo "❌ Backend Build: 실패"
                    throw err
                }
            }
        }
    }

    stage('Prepare Env Files') {
        steps {
            echo "🧩 Prepare Env Files: blockchain/frontend .env 주입"
            script {
                // 디렉토리 보장
                sh 'mkdir -p blockchain frontend'

                // 1) blockchain/.env 주입
                withCredentials([file(credentialsId: 'ENV_BLOCKCHAIN', variabl
e: 'BLOCK_ENV')]) {
                    sh '''
                    install -m 600 -T "$BLOCK_ENV" "blockchain/.env"
                    echo "[ENV] blockchain/.env installed"
                    '''
                }
```

```
                // frontend 빌드용 env
                withCredentials([file(credentialsId: env.BRANCH_NAME == 'mai
n' ? 'FRONT_ENV_PROD' : 'FRONT_ENV_DEV', variable: 'FRONT_BUILD')]) {
                    sh '''
                    install -m 400 -T "$FRONT_BUILD" frontend/.env
                    echo "[ENV] frontend build .env installed"
                    '''
                }

                // frontend 런타임용 env
                withCredentials([file(credentialsId: 'FRONT_ENV_RUNTIME', var
iable: 'FRONT_RUNTIME')]) {
                    sh '''
                    install -m 400 -T "$FRONT_RUNTIME" frontend/.env.runtime
                    echo "[ENV] frontend runtime .env installed"
                    '''
                }
            }
        }
    }

    stage('Hardhat Setup & Compile') {
        when { expression { return env.CHAIN_CHANGED == 'true' } }
        steps {
            echo "⛓️ Hardhat: Node/NPM 설정 및 컴파일"
            dir('blockchain') {
                sh '''#!/usr/bin/env bash
                set -Eeuo pipefail
                echo "[CHAIN] setup start"

                export NVM_DIR="$HOME/.nvm"
                if [ ! -s "$NVM_DIR/nvm.sh" ]; then
                    echo "[CHAIN] installing nvm ..."
                    curl -fsSL https://raw.githubusercontent.com/nvm-sh/nvm/v
0.39.7/install.sh | bash
                fi
                . "$NVM_DIR/nvm.sh"
```

```
                nvm install 20
                nvm use 20

                node -v
                npm -v

                export CI=true
                npm ci --no-audit --no-fund
                npx hardhat compile

                echo "[CHAIN] compile done"
                '''
            }
            echo "✅ Hardhat: 컴파일 완료"
        }
    }

    stage('Deploy to Dev') {
        when { expression { env.BRANCH_NAME == 'dev' } }
        steps {
            echo "🚀 Deploy to Dev: DEV 네트워크/컨테이너 준비"
            script {
                withCredentials([usernamePassword(credentialsId: 'dockerhu
b', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_TOKE
N')]) {
                    sh 'echo "$DOCKER_TOKEN" | docker login -u "$DOCKER_U
SER" --password-stdin'
                }
                // 네트워크가 없으면 생성
                sh "docker network inspect ${TEST_NETWORK} >/dev/null 2>&
1 || docker network create ${TEST_NETWORK}"
                def TAG = sh(script: "git rev-parse --short=12 HEAD", returnStd
out: true).trim()
                env.IMAGE_TAG = TAG

                if (env.BACK_CHANGED == 'true') {
                    echo "📦 DEV Backend: 이미지 빌드 및 컨테이너 실행"
                    script {
```

```
                    try {
                        sh """
                            docker build -f backend/Dockerfile -t majoong/backe
nd-dev:${TAG} backend >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                            docker rm -f ${DEV_BACK_CONTAINER} || true
>> "\$WORKSPACE/${LOG_FILE}" 2>&1
                            docker run -d \
                              --name ${DEV_BACK_CONTAINER} \
                              --network ${TEST_NETWORK} \
                              --network-alias backend-test \
                              -p ${DEV_BACK_PORT}:8080 \
                              majoong/backend-dev:${TAG}
>> "\$WORKSPACE/${LOG_FILE}" 2>&1
                        """
                        echo "✅ DEV Backend: 배포 완료 (tag=${TAG})"
                    } catch(err) {
                        sh "echo '[ERROR] Backend Deploy to Dev failed: ${err}'
>> \"$WORKSPACE/${LOG_FILE}\""
                        echo "❌ DEV Backend: 배포 실패"
                        throw err
                    }
                }
            }

            if (env.FRONT_CHANGED == 'true') {
                echo "🖥️ DEV Frontend: 이미지 빌드 및 컨테이너 실행"
                script {
                    try {
                        sh '''#!/usr/bin/env bash
                        set -Eeuo pipefail

                        # 콘솔과 파일 동시 출력 + 파이프 실패코드 전파
                        set +e
                        set -o pipefail
                        DOCKER_BUILDKIT=1 docker build \
                        --no-cache \
                        --progress=plain \
                        -f frontend/Dockerfile \
```

```
                    --secret id=buildenv,src="$WORKSPACE/frontend/.env"
\
                    -t majoong/frontend-dev:$IMAGE_TAG \
                    frontend 2>&1 | tee -a "$WORKSPACE/${LOG_FILE}"
                    ec=$?
                    set -e

                    echo "[FRONTEND][DEV] docker build exit=$ec" >>
"$WORKSPACE/${LOG_FILE}"
                    exit "$ec"
                    '''



                    // 아래 run 부분은 그대로 두되, 로그 리다이렉션도 이스케이프
권장
                    sh """
                    docker rm -f ${DEV_FRONT_CONTAINER} || true >>
"\$WORKSPACE/\${LOG_FILE}" 2>&1

                    docker run -d \
                      --name ${DEV_FRONT_CONTAINER} \
                      --network ${TEST_NETWORK} \
                      -p ${DEV_FRONT_PORT}:3000 \
                      --env-file "\$WORKSPACE/frontend/.env.runtime" \
                      -v next_cache_dev:/app/.next/cache \
                      --restart unless-stopped \
                      majoong/frontend-dev:\$IMAGE_TAG >> "\$WORKSP
ACE/\${LOG_FILE}" 2>&1
                    """
                    echo "✅ DEV Frontend: 배포 완료 (tag=${TAG})"
                  } catch (err) {
                    sh "echo '[ERROR] Frontend Deploy to Dev failed: ${er
r}' >> \"$WORKSPACE/${LOG_FILE}\""
                    echo "❌ DEV Frontend: 배포 실패"
                    throw err
                  }
                }
              }
```

```
            }
          }
        }

        stage('Deploy to Prod') {
          when { expression { env.BRANCH_NAME == 'main' } }
          steps {
            echo "🚀 Deploy to Prod: PROD 네트워크/컨테이너 준비"
            script {
              withCredentials([usernamePassword(credentialsId: 'dockerhu
b', usernameVariable: 'DOCKER_USER', passwordVariable: 'DOCKER_TOKE
N')]) {
                sh 'echo "$DOCKER_TOKEN" | docker login -u "$DOCKER_U
SER" --password-stdin'
              }
              sh "docker network inspect ${PROD_NETWORK} >/dev/null 2>
&1 || docker network create ${PROD_NETWORK}"
              def TAG = sh(script: "git rev-parse --short=12 HEAD", returnStd
out: true).trim()
              env.IMAGE_TAG = TAG

             if (env.BACK_CHANGED == 'true') {
                echo "📦 PROD Backend: 이미지 빌드/태깅 및 컨테이너 실행"
                script {
                  try {
                    sh """
                      docker build -f backend/Dockerfile -t majoong/backe
nd-prod:${TAG} backend  >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                      docker tag majoong/backend-prod:${TAG} majoong/
backend-prod:latest        >> "\$WORKSPACE/${LOG_FILE}" 2>&1
                      docker rm -f ${PROD_BACK_CONTAINER} || true
>> "\$WORKSPACE/${LOG_FILE}" 2>&1
                      docker run -d \
                        --name ${PROD_BACK_CONTAINER} \
                        --network ${PROD_NETWORK} \
                        --network-alias backend \
                        -p ${PROD_BACK_PORT}:8080 \
                        majoong/backend-prod:${TAG}
```

```
>> "\$WORKSPACE/${LOG_FILE}" 2>&1
                    """
                    echo "✅ PROD Backend: 배포 완료 (tag=${TAG})"
                } catch(err) {
                    sh "echo '[ERROR] Backend Deploy to main failed: ${er
r}' >> \"$WORKSPACE/${LOG_FILE}\""
                    echo "❌ PROD Backend: 배포 실패"
                    throw err
                }
            }
        }

        if (env.FRONT_CHANGED == 'true') {
            echo "🖥️ PROD Frontend: 이미지 빌드/태깅 및 컨테이너 실행"
            script {
                try {
                    sh '''#!/usr/bin/env bash
                    set -Eeuo pipefail

                    set +e
                    set -o pipefail
                    DOCKER_BUILDKIT=1 docker build \
                    --no-cache \
                    --progress=plain \
                    -f frontend/Dockerfile \
                    --secret id=buildenv,src="$WORKSPACE/frontend/.env"
\
                    -t majoong/frontend-prod:$IMAGE_TAG \
                    frontend 2>&1 | tee -a "$WORKSPACE/${LOG_FILE}"
                    ec=$?
                    set -e

                    echo "[FRONTEND][PROD] docker build exit=$ec" >>
"$WORKSPACE/${LOG_FILE}"
                    docker tag majoong/frontend-prod:$IMAGE_TAG majoon
g/frontend-prod:latest >> "$WORKSPACE/${LOG_FILE}" 2>&1 || true
                    exit "$ec"
                    '''
```

```
                    sh """
                    docker rm -f ${PROD_FRONT_CONTAINER} || true >>
"\$WORKSPACE/\${LOG_FILE}" 2>&1

                    docker run -d \
                      --name ${PROD_FRONT_CONTAINER} \
                      --network ${PROD_NETWORK} \
                      -p ${PROD_FRONT_PORT}:3000 \
                      --env-file "\$WORKSPACE/frontend/.env.runtime" \
                      -v next_cache_prod:/app/.next/cache \
                      --restart unless-stopped \
                      majoong/frontend-prod:\$IMAGE_TAG >> "\$WORKS
PACE/\${LOG_FILE}" 2>&1

                    """

                    echo "✅ PROD Frontend: 배포 완료 (tag=${TAG})"
                  } catch(err) {
                    sh "echo '[ERROR] Frontend Deploy to Main failed: ${er
r}' >> \"$WORKSPACE/${LOG_FILE}\""
                    echo "❌ PROD Frontend: 배포 실패"
                    throw err
                  }
                }
              }
            }
          }
        }
      }

    post {
      success {
        echo "🎉 POST: 빌드 성공 – Mattermost 알림 전송"
        script {
          def branch   = resolveBranch()
          def mention  = resolvePusherMention()       // @username 또는
```

```
빈 문자열
            def commitMsg = sh(script: "git log -1 --pretty=%s", returnStdout:
true).trim()
            def commitUrl = env.GIT_COMMIT_URL ?: ""
            sendMMNotify(true, [
               branch   : branch,
               mention  : mention,
               buildUrl : env.BUILD_URL,
               commit   : [msg: commitMsg, url: commitUrl],
               // 실패가 아니므로 details 생략
            ])
         }
      }
      failure {
         echo "🚨 POST: 빌드 실패 – 로그 tail 후 Mattermost 알림 전송"
         script {
            def branch    = resolveBranch()
            def mention   = resolvePusherMention()
            def commitMsg = sh(script: "git log -1 --pretty=%s", returnStdout:
true).trim()
            def commitUrl = env.GIT_COMMIT_URL ?: ""

            // ci.log이 있으면 마지막 200줄, 없으면 빈 문자열
            def tail = sh(
               script: "tail -n 150 \"$WORKSPACE/${LOG_FILE}\" 2>/dev/null ||
true",
               returnStdout: true
            ).trim()

            // (선택) 민감정보 간단 마스킹
            tail = tail
               .replaceAll(/(?i)(token|secret|password|passwd|apikey|api_key)
\\s*[:=]\\s*\\S+/, '$1=[REDACTED]')
               .replaceAll(/AKIA[0-9A-Z]{16}/, 'AKIA[REDACTED]')

            def detailsBlock = tail ? "```text\n${tail}\n```" : ""

            sendMMNotify(false, [
```

```
                branch   : branch,
                mention  : mention,
                buildUrl : env.BUILD_URL,
                commit   : [msg: commitMsg, url: commitUrl],
                details  : detailsBlock
            ])
        }
      }
      always {
        echo "📦 Pipeline finished with status: ${currentBuild.currentResult} – 🔥 민감 파일 정리"
        sh "rm -f ${env.BACKEND_DIR}/src/main/resources/application.yml || true"
        // ⬇️ runtime 파일까지 함께 제거
        sh "rm -f blockchain/.env frontend/.env frontend/.env.runtime || true"
        echo "🧹 Cleanup: application.yml/.env 삭제 완료"
      }
    }
}

// 브랜치 해석: BRANCH_NAME → GIT_REF → git
def resolveBranch() {
  if (env.BRANCH_NAME) return env.BRANCH_NAME
  if (env.GIT_REF) return env.GIT_REF.replaceFirst(/^refs\/heads\//,'')
  return sh(script: "git name-rev --name-only HEAD || git rev-parse --abbrev-ref HEAD", returnStdout: true).trim()
}

// @username (웹훅의 user_username) 우선, 없으면 커밋 작성자 표시
def resolvePusherMention() {
  def u = env.GIT_PUSHER_USERNAME?.trim()
  if (u) return "@${u}"
  return sh(script: "git --no-pager show -s --format='%an <%ae>' HEAD", returnStdout: true).trim()
}

// ✅/❌ 제목을 "## :jenkins7: Jenkins Build Success ✅ / Failed ❌" 로 출력
```

하고

```
// 아래에 pusher / Target Branch / Commit (실패 시 Error)만 표시
def sendMMNotify(boolean success, Map info) {
  def titleLine = success ? "## :jenkins7: Jenkins Build Success"
                          : "## :angry_jenkins: Jenkins Build Failed"

  def lines = []
  if (info.mention) lines << "**Author**: ${info.mention}"
  if (info.branch)  lines << "**Target Branch**: `${info.branch}`"
  if (info.commit?.msg) {
    def commitLine = info.commit?.url ? "[${info.commit.msg}](${info.commi
t.url})" : info.commit.msg
    lines << "**Commit**: ${commitLine}"
  }
  if (!success && info.details) {
    lines << "**Error Message**:\n${info.details}"
  }

  def text = "${titleLine}\n" + (lines ? ("\n" + lines.join("\n")) : "")

  // 안전 전송(크리덴셜 경고 없음)
  writeFile file: 'payload.json', text: groovy.json.JsonOutput.toJson([
    text      : text,
    username  : "Jenkins",
    icon_emoji: ":jenkins7:"
  ])
  withCredentials([string(credentialsId: 'mattermost-webhook', variable: 'M
M_WEBHOOK')]) {
    sh(script: '''
      curl -sS -f -X POST -H 'Content-Type: application/json' \
        --data-binary @payload.json \
        "$MM_WEBHOOK"
    ''')
  }
}
```

# 4. nginx - majoong.conf

**/etc/nginx/conf.d 하위에 폴더 만들어서 아래 파일 넣기**

```
# ---------- 업스트림 ----------
upstream fe_prod { server 127.0.0.1:3000;  keepalive 32; }
upstream be_prod { server 127.0.0.1:8082;  keepalive 32; }
upstream fe_test { server 127.0.0.1:3001;  keepalive 16; }
upstream be_test { server 127.0.0.1:8081;  keepalive 16; }

# ---------- 공통 헤더/웹소켓 ----------
map $http_upgrade $connection_upgrade { default upgrade; '' close; }

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 X-Forwarded-Host  $host;
proxy_set_header X-Forwarded-Port  $server_port;
# ---------- 잘못된 Host/IP 직접접속 차단 ----------
server { listen 80  default_server;  return 444; }
server {
  listen 443 ssl http2 default_server;
  # 아무 인증서나(혹은 와일드카드) 지정 가능하지만 어차피 444 반환
  ssl_certificate     /etc/letsencrypt/live/majoong.site/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/majoong.site/privkey.pem;
  return 444;
}

# ================================================================
# 1) 운영 프론트: majoong.site, www.majoong.site
# ================================================================
server {
  listen 80;
  server_name majoong.site www.majoong.site;
  return 301 https://$host$request_uri;
}
```

```
server {
  listen 443 ssl http2;
  server_name majoong.site www.majoong.site;

  ssl_certificate     /etc/letsencrypt/live/majoong.site/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/majoong.site/privkey.pem;

  client_max_body_size 50m;

  location / {
    proxy_read_timeout 300s;
    proxy_send_timeout 300s;
    proxy_pass http://fe_prod;
  }

  location /api/ {
    proxy_pass http://be_prod;
  }

  location /oauth2/ {
    proxy_pass http://be_prod;
  }

  location /login/oauth2/ {
    proxy_pass http://be_prod;
  }

}

# ================================================================
# 2) 운영 백엔드: api.majoong.site  (루트부터 백엔드!)
# ================================================================
server {
  listen 80;
  server_name api.majoong.site;
  return 301 https://$host$request_uri;
```

```
}
server {
  listen 443 ssl http2;
  server_name api.majoong.site;

  ssl_certificate     /etc/letsencrypt/live/api.majoong.site/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/api.majoong.site/privkey.pem;

  client_max_body_size 50m;

  location / {
    proxy_read_timeout 300s;
    proxy_send_timeout 300s;
    proxy_pass http://be_prod;
  }

  location /swagger-ui/ {
    proxy_read_timeout 120s;
    proxy_pass http://be_prod;
  }

  location ^~ /v3/api-docs {
    proxy_read_timeout 120s;
    proxy_pass http://be_prod;
  }

  location = /healthz      { access_log off; proxy_pass http://be_prod/actuato
r/health; }

  location = / { return 302 /swagger-ui/index.html; }

  location /login/oauth2/ {
    proxy_pass http://be_prod;
  }
}

# ================================================
==
```

```
# 3) 테스트 프론트: test.majoong.site
# ================================================================
server {
  listen 80;
  server_name test.majoong.site;
  return 301 https://$host$request_uri;
}
server {
  listen 443 ssl http2;
  server_name test.majoong.site;

  ssl_certificate     /etc/letsencrypt/live/test.majoong.site/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/test.majoong.site/privkey.pem;

  client_max_body_size 50m;

  location / {
    proxy_read_timeout 300s;
    proxy_send_timeout 300s;
    proxy_pass http://fe_test;
  }

  location /api/ {
    proxy_pass http://be_test;
  }

  location /oauth2/ {
    proxy_pass http://be_test;
  }

  location /login/oauth2/ {
    proxy_pass http://be_test;
  }


}
```

```
# ================================================================
# 4) 테스트 백엔드: api-test.majoong.site
# ================================================================
server {
  listen 80;
  server_name api-test.majoong.site;
  return 301 https://$host$request_uri;
}
server {
  listen 443 ssl http2;
  server_name api-test.majoong.site;

  ssl_certificate     /etc/letsencrypt/live/api-test.majoong.site/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/api-test.majoong.site/privkey.pem;

  client_max_body_size 50m;

  location /              {
    proxy_read_timeout 300s;
    proxy_send_timeout 300s;
    proxy_pass http://be_test;
  }

  location /swagger-ui/    {
    proxy_read_timeout 120s;
    proxy_pass http://be_test;
  }

  location ^~ /v3/api-docs {
    proxy_read_timeout 120s;
    proxy_pass http://be_test;
  }

  location = /healthz      { access_log off; proxy_pass http://be_test/actuator/health; }
```

```
  location = / { return 302 /swagger-ui/index.html; }

  location /login/oauth2/ {
    proxy_pass http://be_test;
  }
}
# ========================================================
==
# 5) 싸피domain: j13e105.p.ssafy.io
# ========================================================
==
# j13e105 단일 도메인
server {
  listen 80;
  server_name j13e105.p.ssafy.io;
  return 301 https://www.majoong.site$request_uri;
}

server {
  listen 443 ssl http2;
  server_name j13e105.p.ssafy.io;

  ssl_certificate     /etc/letsencrypt/live/j13e105.p.ssafy.io/fullchain.pem;
  ssl_certificate_key /etc/letsencrypt/live/j13e105.p.ssafy.io/privkey.pem;

  client_max_body_size 50m;

  # -------- 백엔드로 보낼 경로들 --------
  # Swagger UI
  location ^~ /swagger-ui/     { proxy_read_timeout 120s; proxy_pass http://b
e_prod; }
  # Swagger 리소스(필요 시)
  location ^~ /swagger-resources/ { proxy_read_timeout 120s; proxy_pass ht
tp://be_prod; }
  location ^~ /webjars/        { proxy_read_timeout 120s; proxy_pass http://be
_prod; }
  # OpenAPI JSON
  location ^~ /v3/api-docs     { proxy_read_timeout 120s; proxy_pass http://b
```

```
e_prod; }
  # 실제 API
  location ^~ /api/           { proxy_read_timeout 300s; proxy_send_timeout 30
0s; proxy_pass http://be_prod; }

  # 루트로 오면 Swagger 바로 보고 싶다면(선택)
  # location = / { return 302 /swagger-ui/index.html; }

  # -------- 나머지는 프론트 --------
  location / {
    proxy_read_timeout 300s;
    proxy_send_timeout 300s;
    proxy_pass http://fe_prod;
  }

  return 301 https://www.majoong.site$request_uri;
}
```