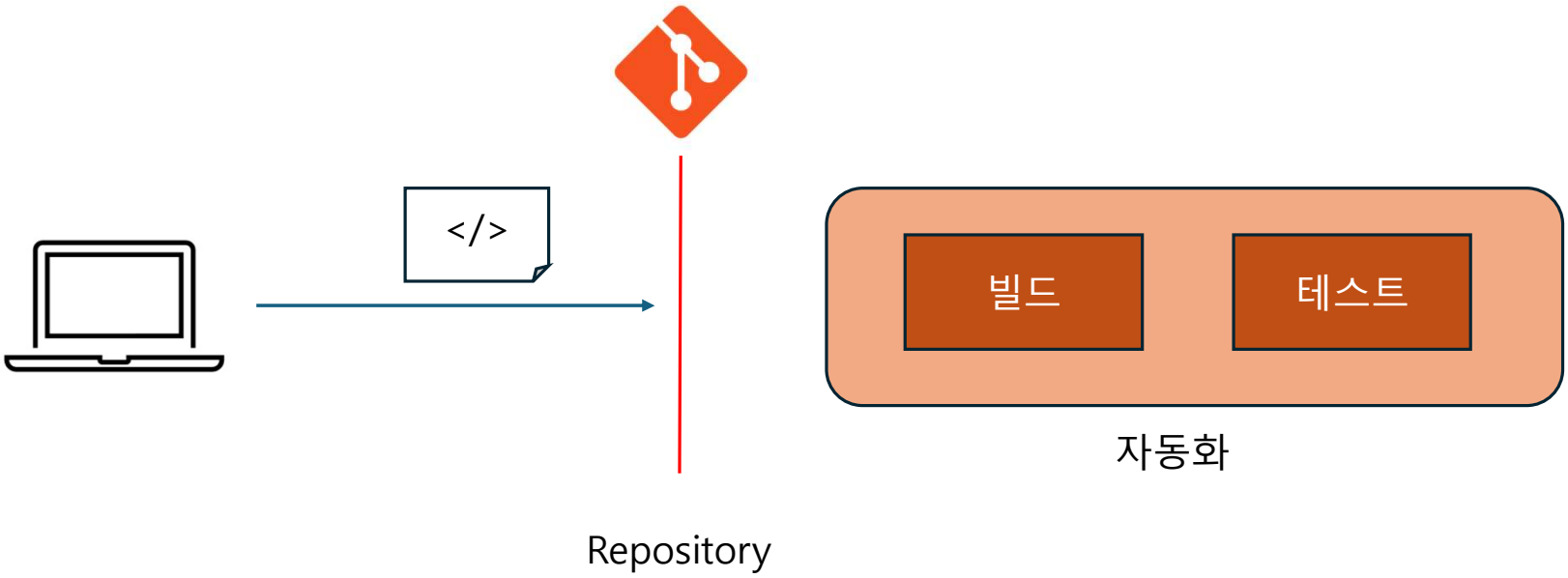
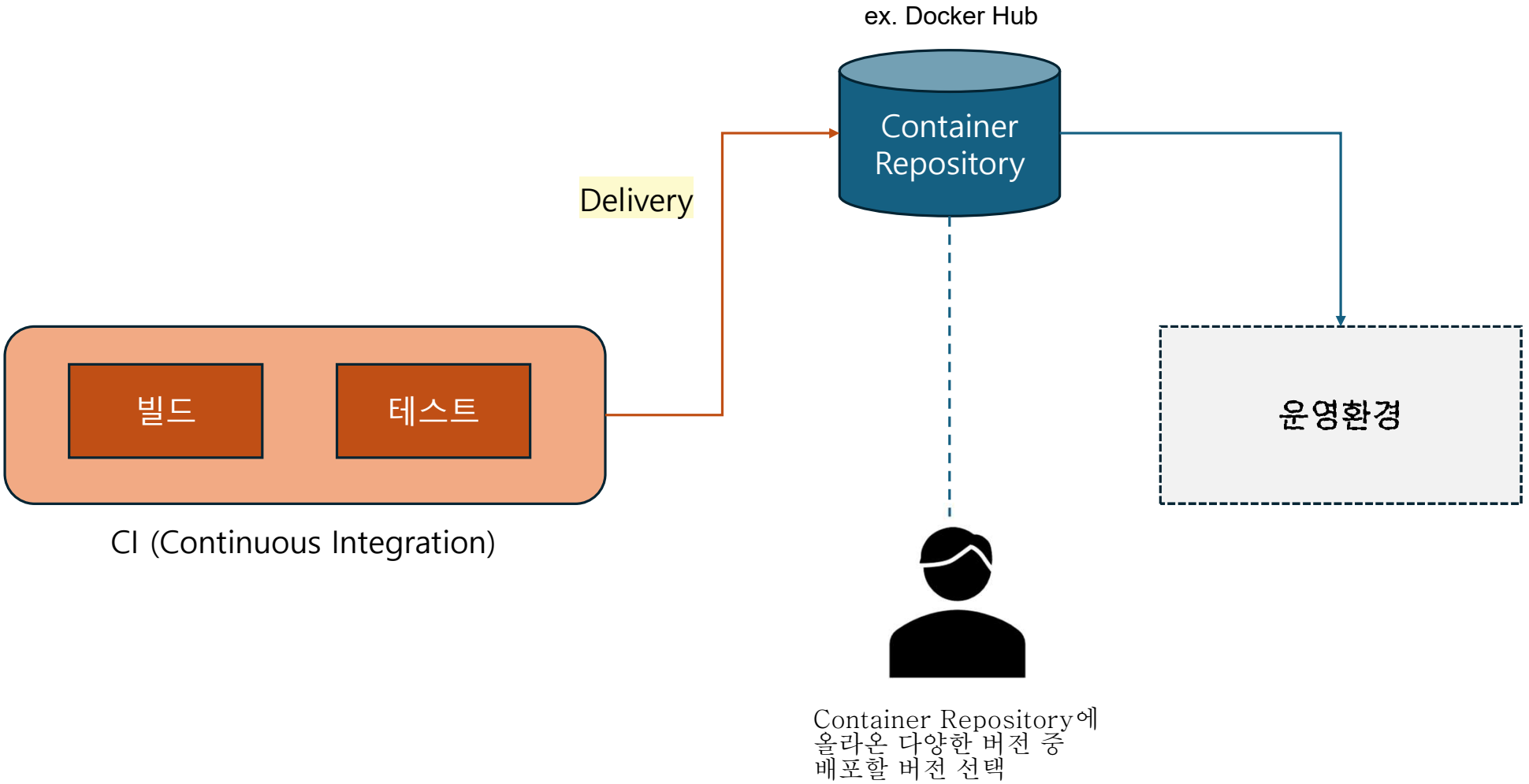
CI/CD
Jenkins

CI (Continuous Integration)

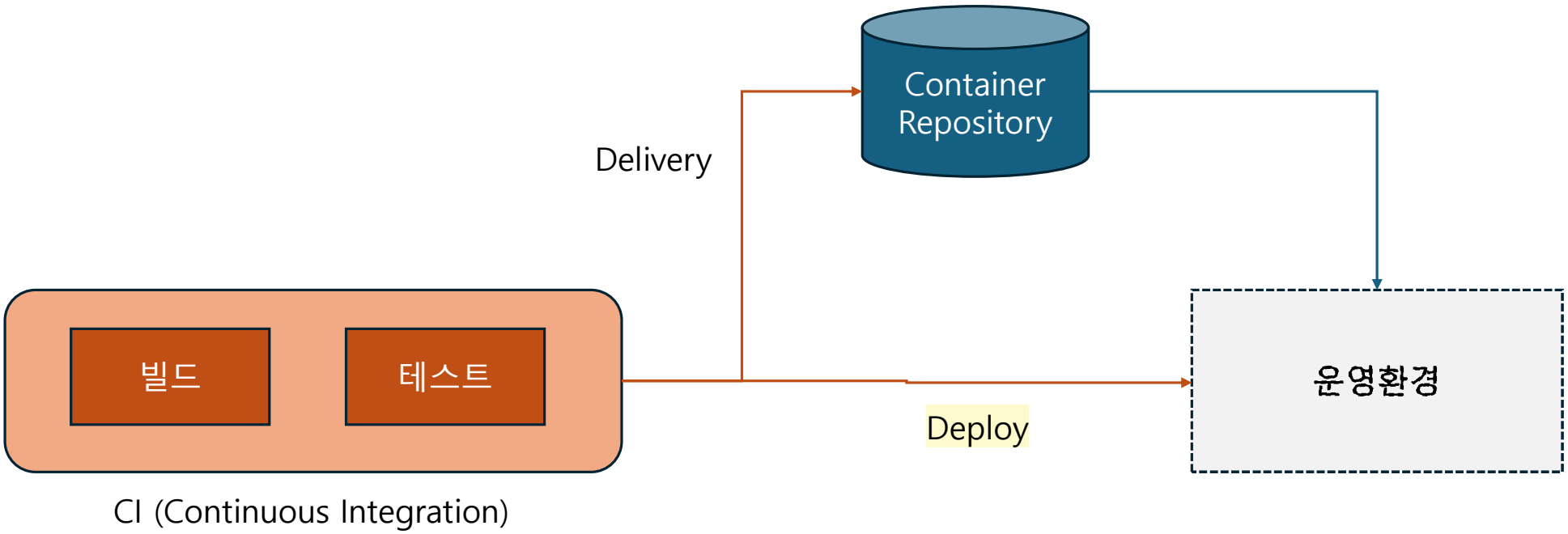
- 개발자는 코드 변경사항을 주기적으로 빈번하게 머지해야 한다.
- 빌드/테스트의 자동화



CD (Continuous Delivery)



CD (Continuous Deploy)



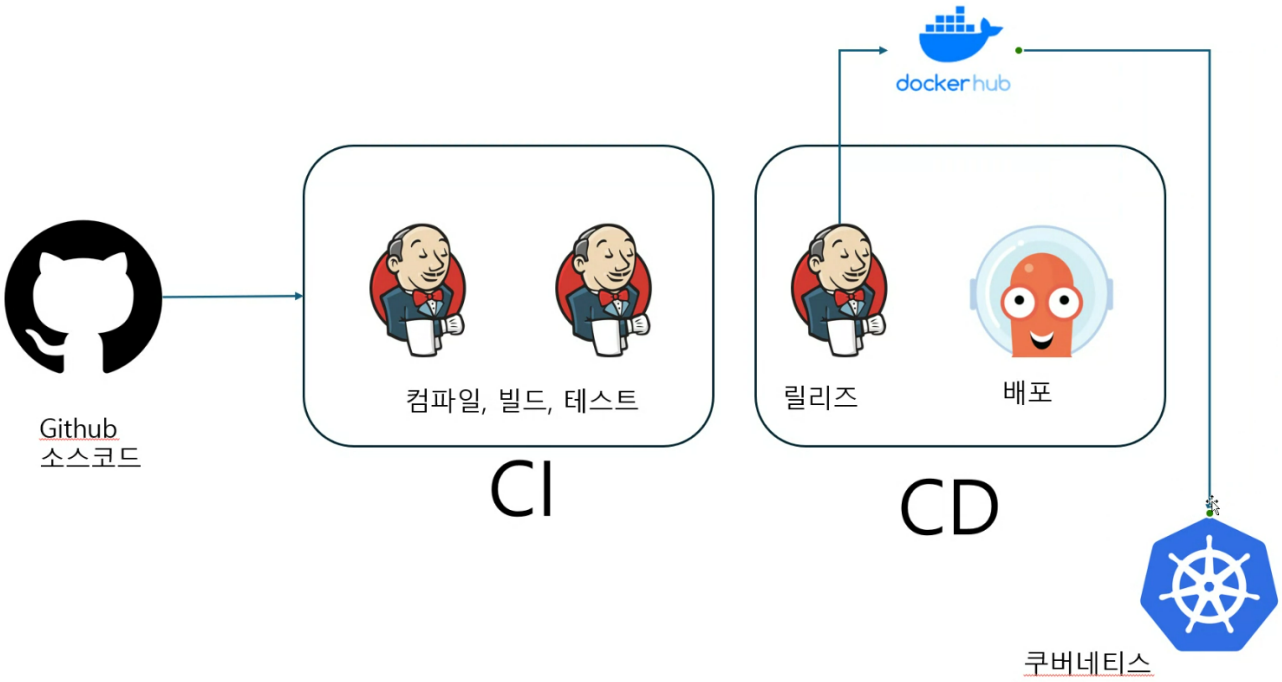
+ Build, Test, Delivery : Jenkins
+ Deploy : ArgoCD + Kubernetes Deploy

Jenkins 소개

- 소프트웨어 개발 프로세스를 자동화하는데 사용되는 오픈 소스 자동화 플랫폼
- 주로 CI/CD(Continuous Integration/Continuous Delivery) 파이프라인 구축에 활용

주요 기능

- 자동화된 빌드 및 테스트
- CI/CD 파이프라인 구축 (delivery)
- 다양한 플러그인 지원



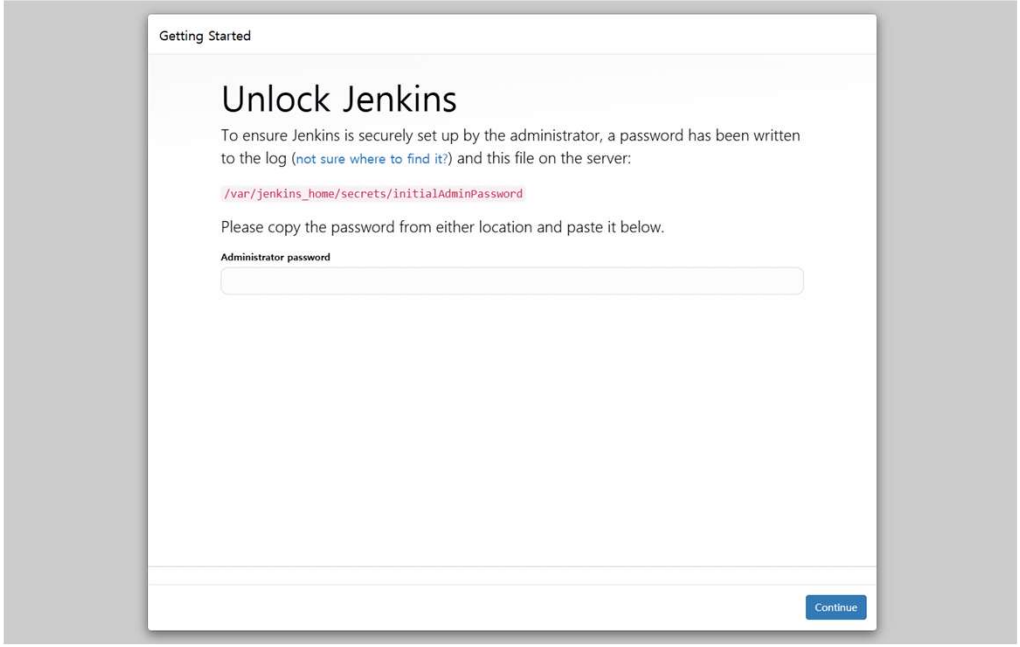
Install Suggested Plugins

```yaml
version: "3"
services:
  jenkins:
    image: solarhc/jenkins:jdk17
    container_name: jenkins
    restart: always
    user: root
    volumes:
      - ./jenkins_home:/var/jenkins_home
      - /var/run/docker.sock:/var/run/docker.sock
    ports:
      - 8000:8080
    environment:
      - DOCKER_HOST=unix:///var/run/docker.sock
```

docker-compose.yml

- docker-compose.yml 파일을 C:₩server₩Jenkins 디렉토리에 생성
- C:₩server₩Jenkins 디렉토리에 jenkins_home 디렉토리 생성
  - C:₩server₩jenkins> **mkdir jenkins_home**
- C:₩server₩Jenkins 디렉토리에서 docker compose up 명령어 실행
  - C:₩server₩Jenkins> **docker compose up -d**

http://localhost:8000 접속



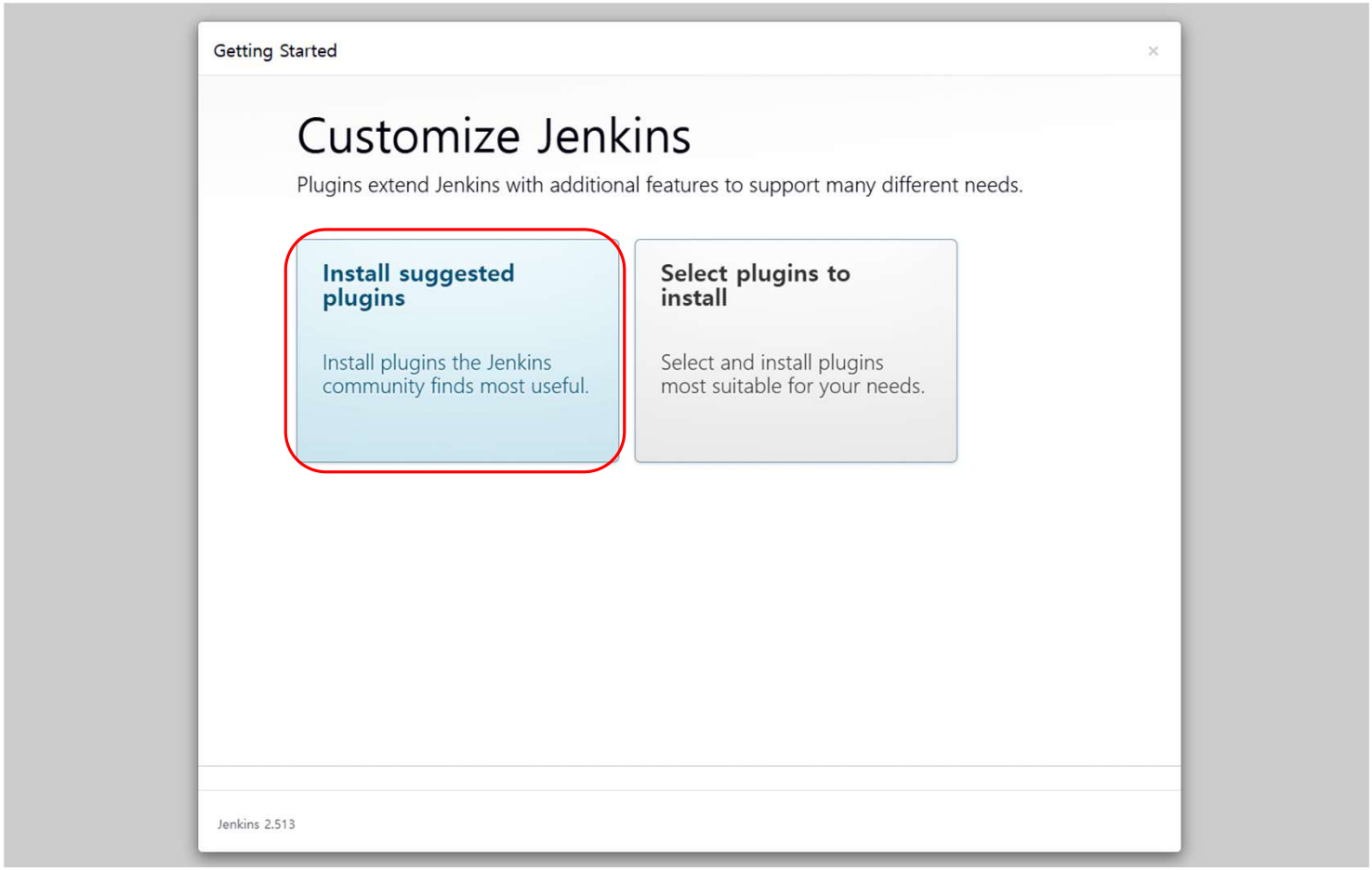
초기 비밀번호는 C:₩server₩jenkins₩jenkins_home₩secrets 디렉토리 initialAdminPassword 파일에 있음.

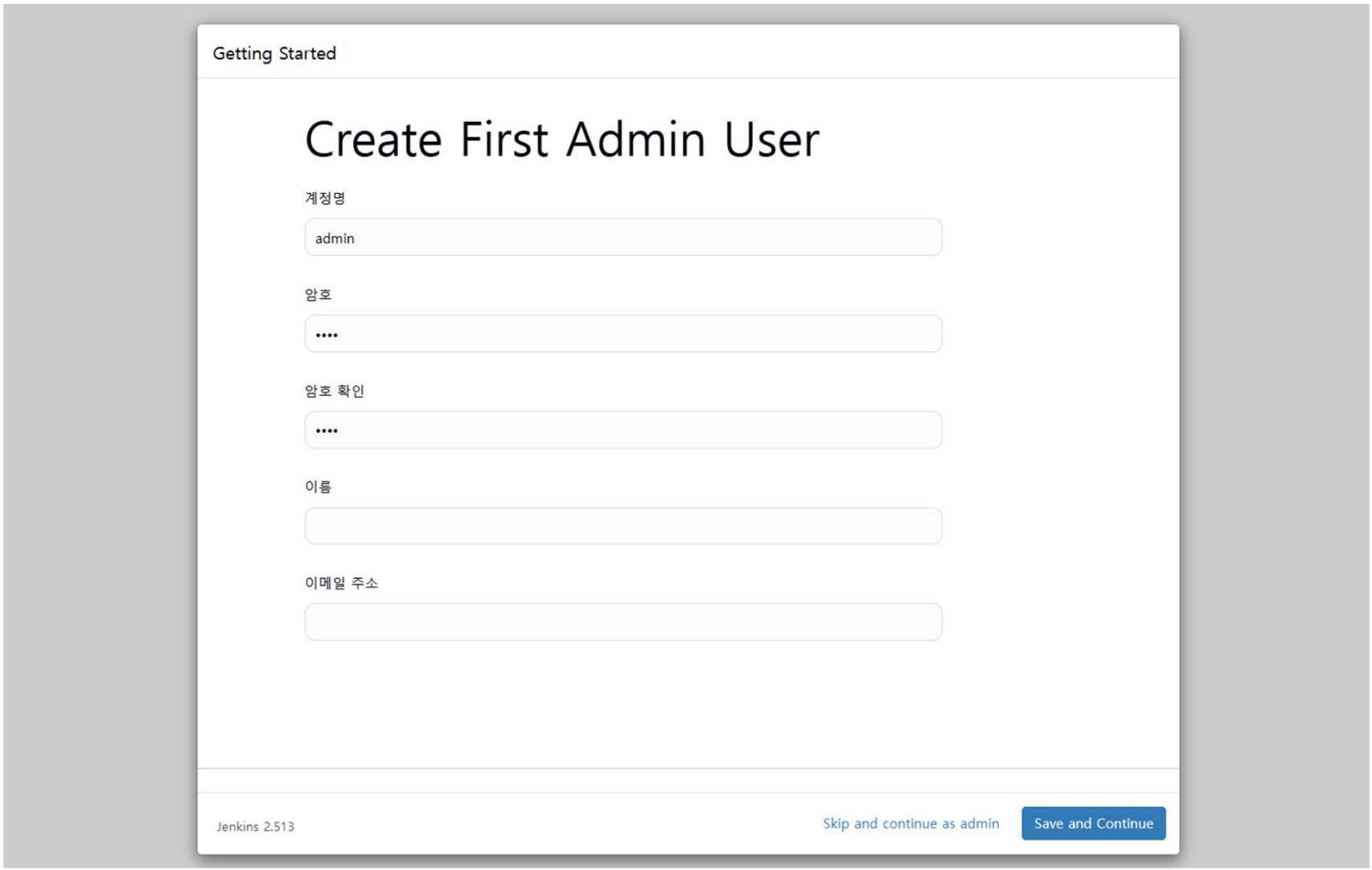
C:₩server₩jenkins₩jenkins_home₩secrets> **cat initialAdminPassword**

2adb81754e2440f886113c5218b75066

초기 비밀번호
출력 명령어

Install Suggested Plugins




Getting Started
Create First Admin User
계정명
admin
암호
암호 확인
이름
이메일 주소
Jenkins 2.513
Skip and continue as admin
Save and Continue

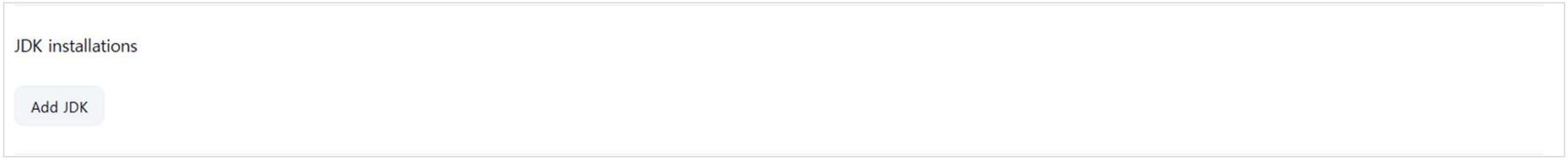
추가 Plugin 설치

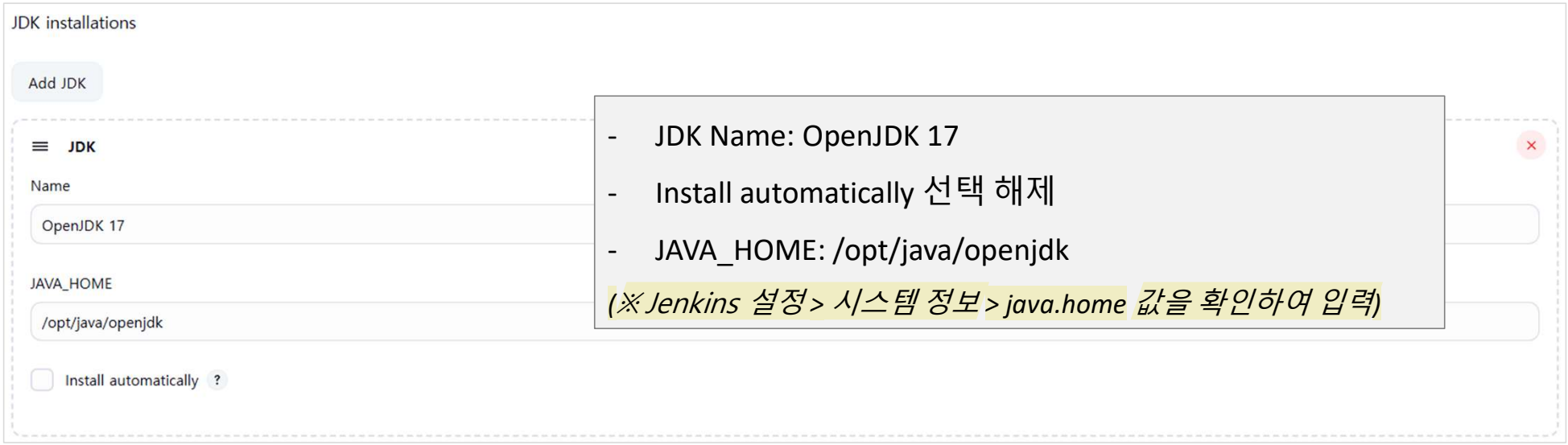
'Jenkins 관리 > 플러그인 관리' 에서 다음 플러그인들을 설치

- Authentication Tokens API ← Jenkins API 토큰을 생성하고 관리하는 기능
- Docker ← Jenkins가 Docker 컨테이너를 관리하고 실행할 수 있도록 통합 기능
- Docker Pipeline ← Jenkins 파이프라인에서 Docker 기능을 더 쉽게 사용할 수 있도록 돕는 플러그인
- Generic Webhook Trigger ← 외부 서비스(예: GitHub, GitLab 등)에서 특정 이벤트가 발생했을 때 Jenkins 빌드를 자동으로 트리거할 수 있도록 Webhook을 설정하는 기능
- Git Parameter ← Jenkins 빌드 시 Git 저장소의 브랜치, 태그 또는 커밋 해시 등을 파라미터로 선택할 수 있도록 함
- Stage View ← Jenkins 파이프라인의 각 '단계(Stage)' 진행 상황을 시각적으로 보여주는 플러그인

Global Tool 설정 - JDK

Jenkins 관리 > Tools



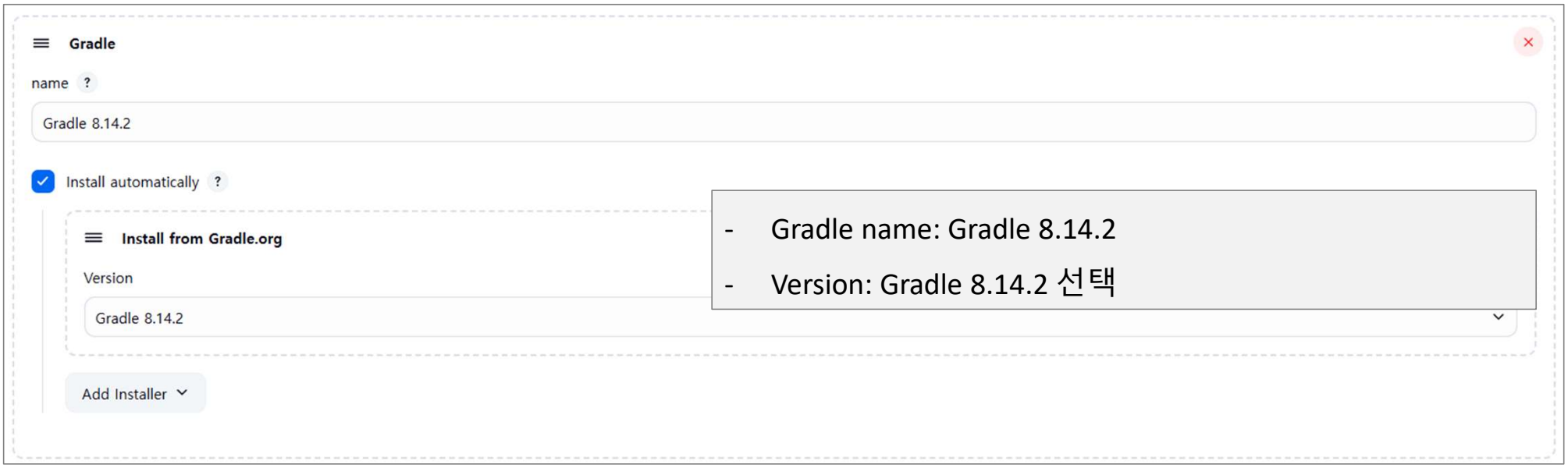


- JDK Name: OpenJDK 17
- Install automatically 선택 해제
- JAVA_HOME: /opt/java/openjdk

(※Jenkins 설정 > 시스템 정보 > java.home 값을 확인하여 입력)

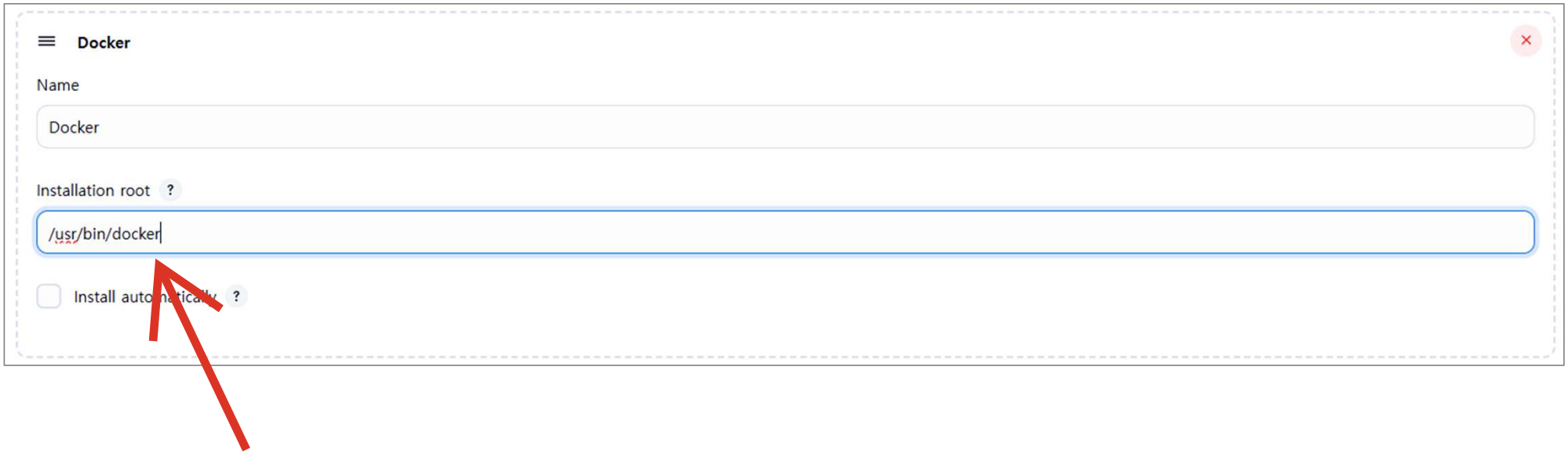
Global Tool 설정 – Gradle

Jenkins 관리 > Tools



- Gradle name: Gradle 8.14.2
- Version: Gradle 8.14.2 선택

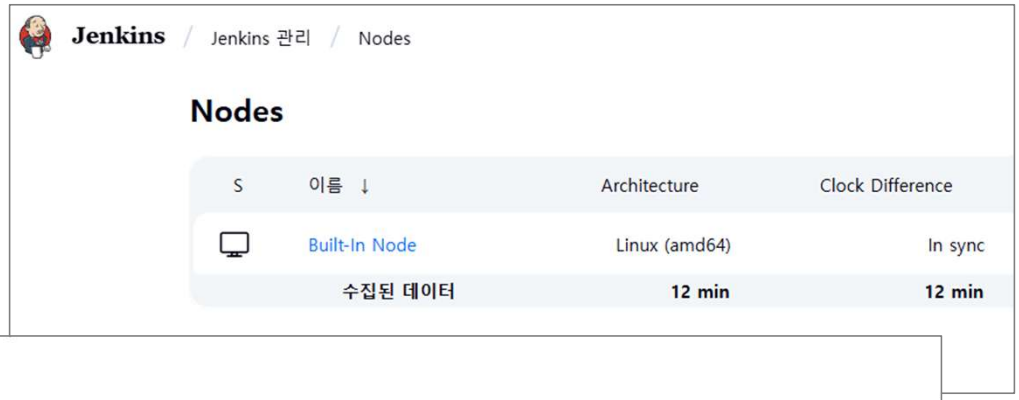
Global Tool 설정 – Docker

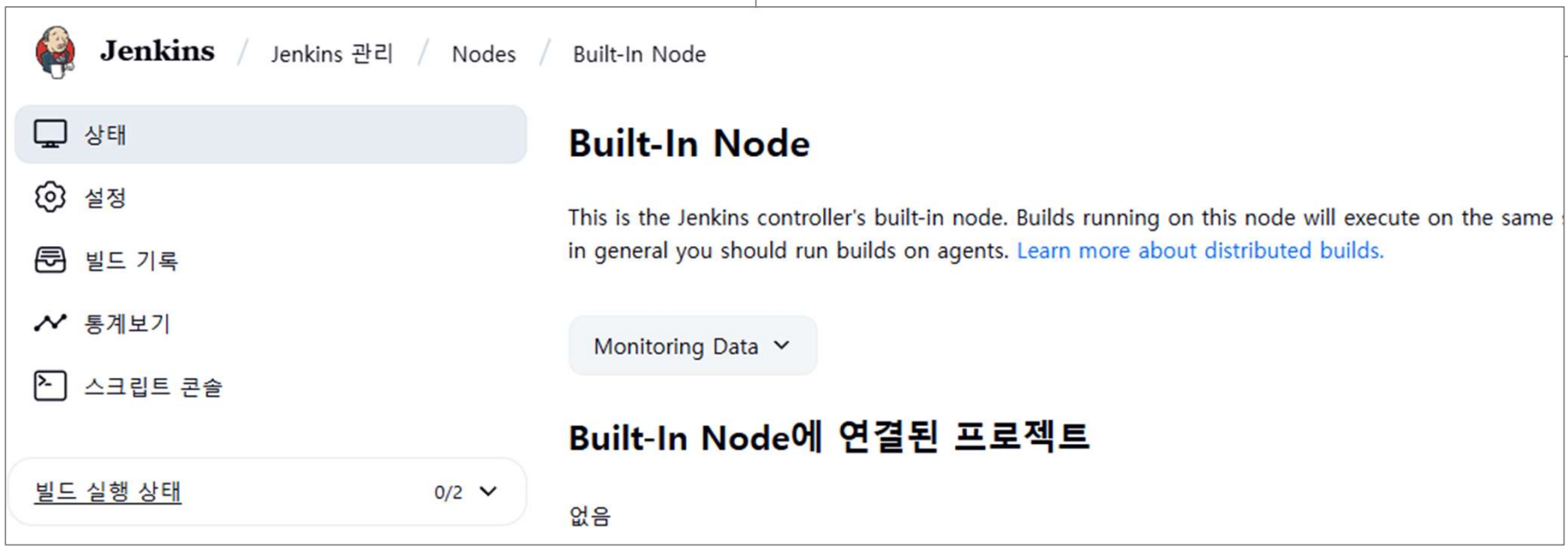


1. docker ps | findstr jenkins : jenkins 컨테이너 찾기
2. docker exec –it (container hash value) bash : Docker Bash 사용 (ex. 컨테이너 내의 OS 환경으로 접속)
3. which docker : docker 폴더 혹은 파일이 위치한 경로 찾기

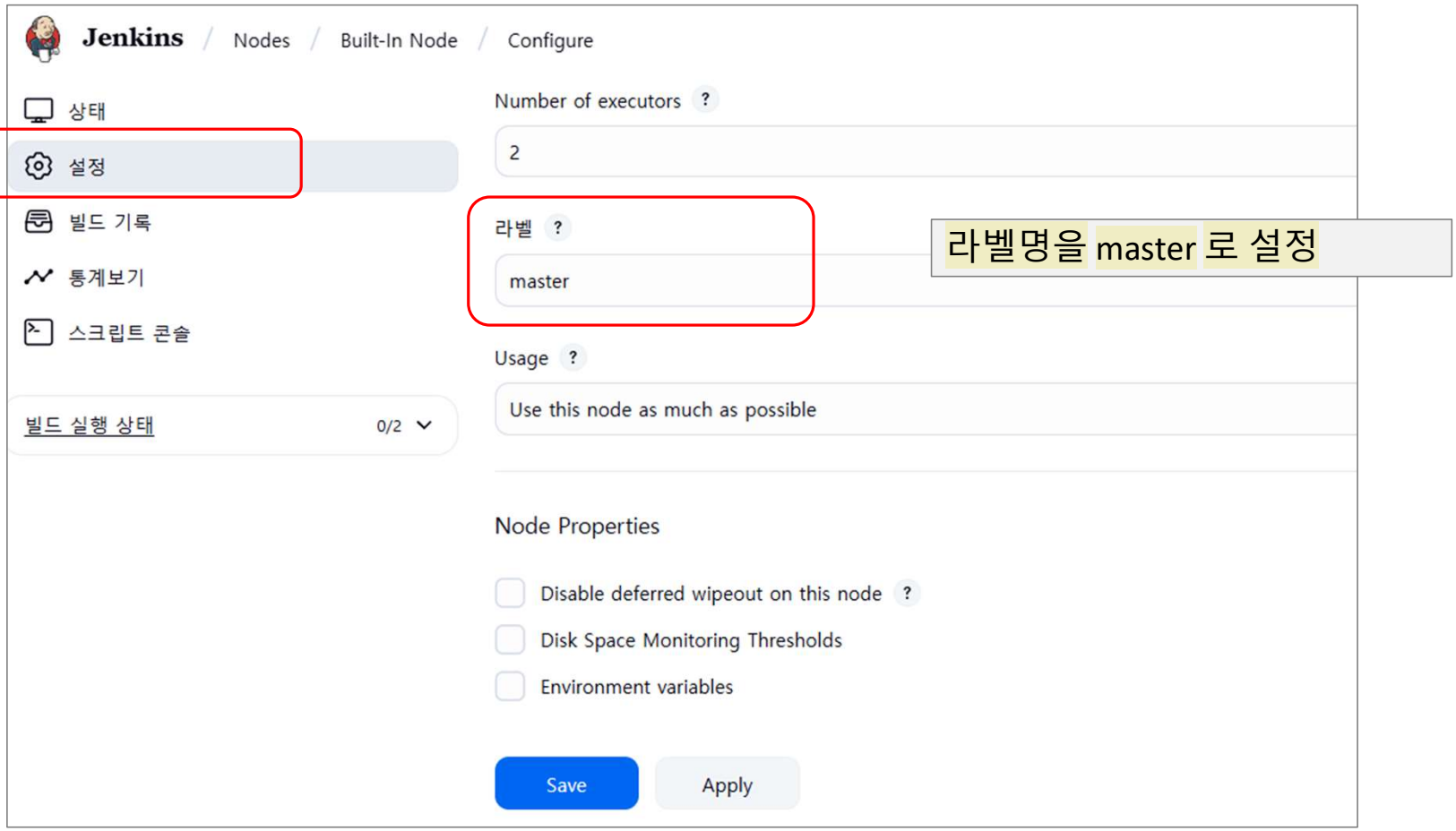
노드 레이블 설정

Jenkins 관리 > Nodes > Build-In Node

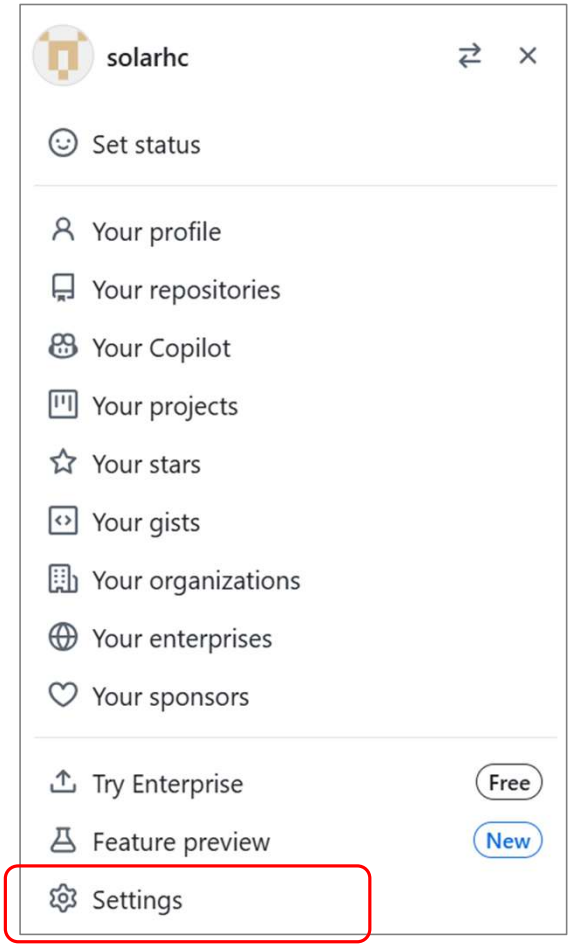




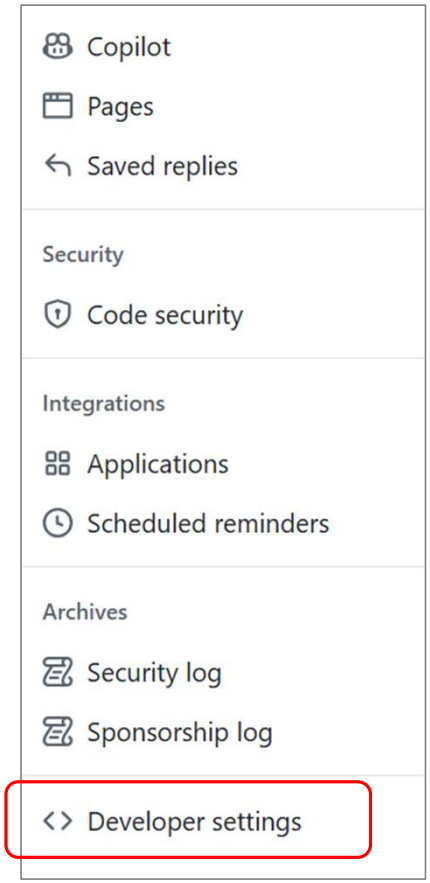
노드 레이블 설정 (계속)

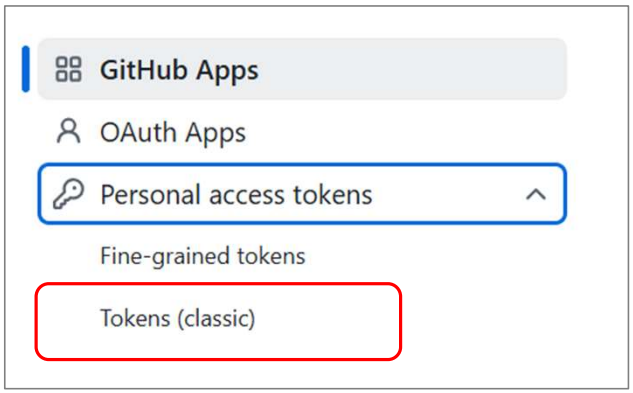

Jenkins / Nodes / Built-In Node / Configure
상태
설정
빌드 기록
통계보기
스크립트 콘솔
빌드 실행 상태 0/2
Number of executors ?
2
라벨 ?
master
Usage ?
Use this node as much as possible
Node Properties
Disable deferred wipeout on this node ?
Disk Space Monitoring Thresholds
Environment variables
Save
Apply
라벨명을 master 로 설정

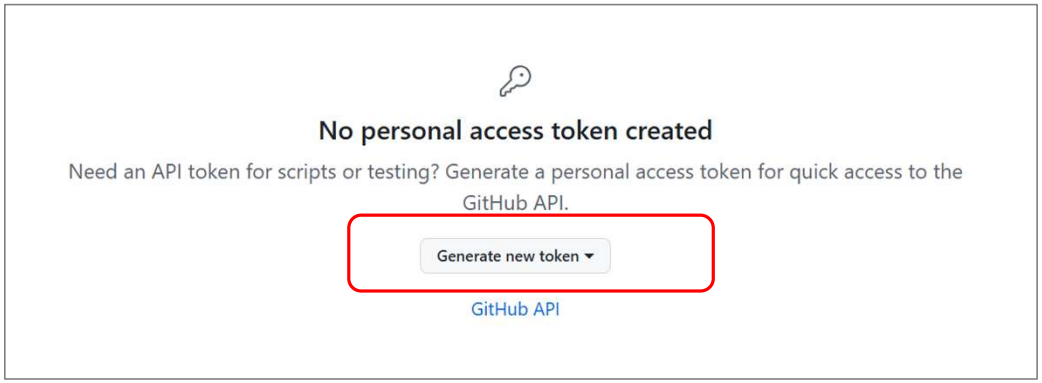
Credential 등록 – github access token 발급



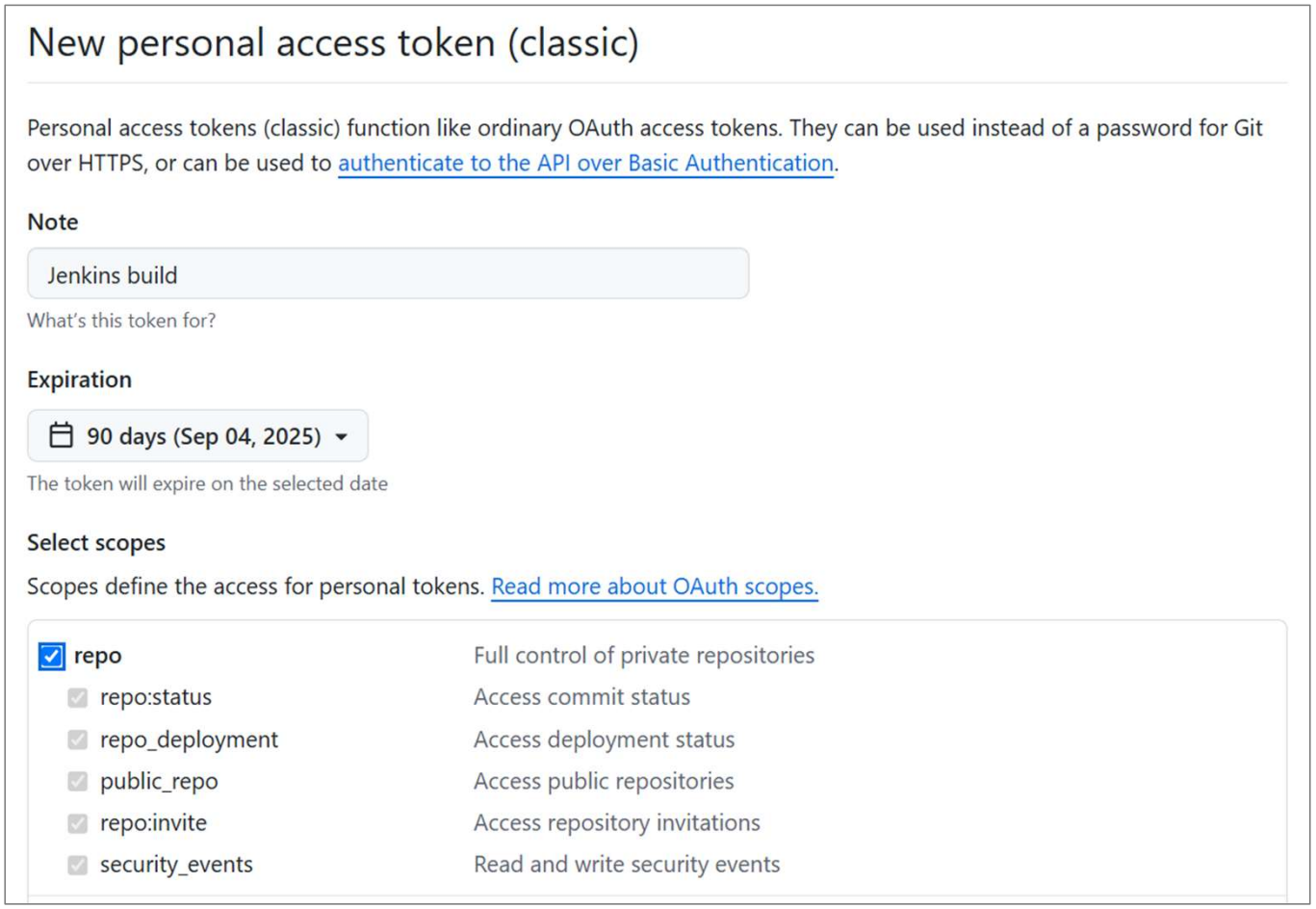
Credential 등록 – github access token 발급 (계속)



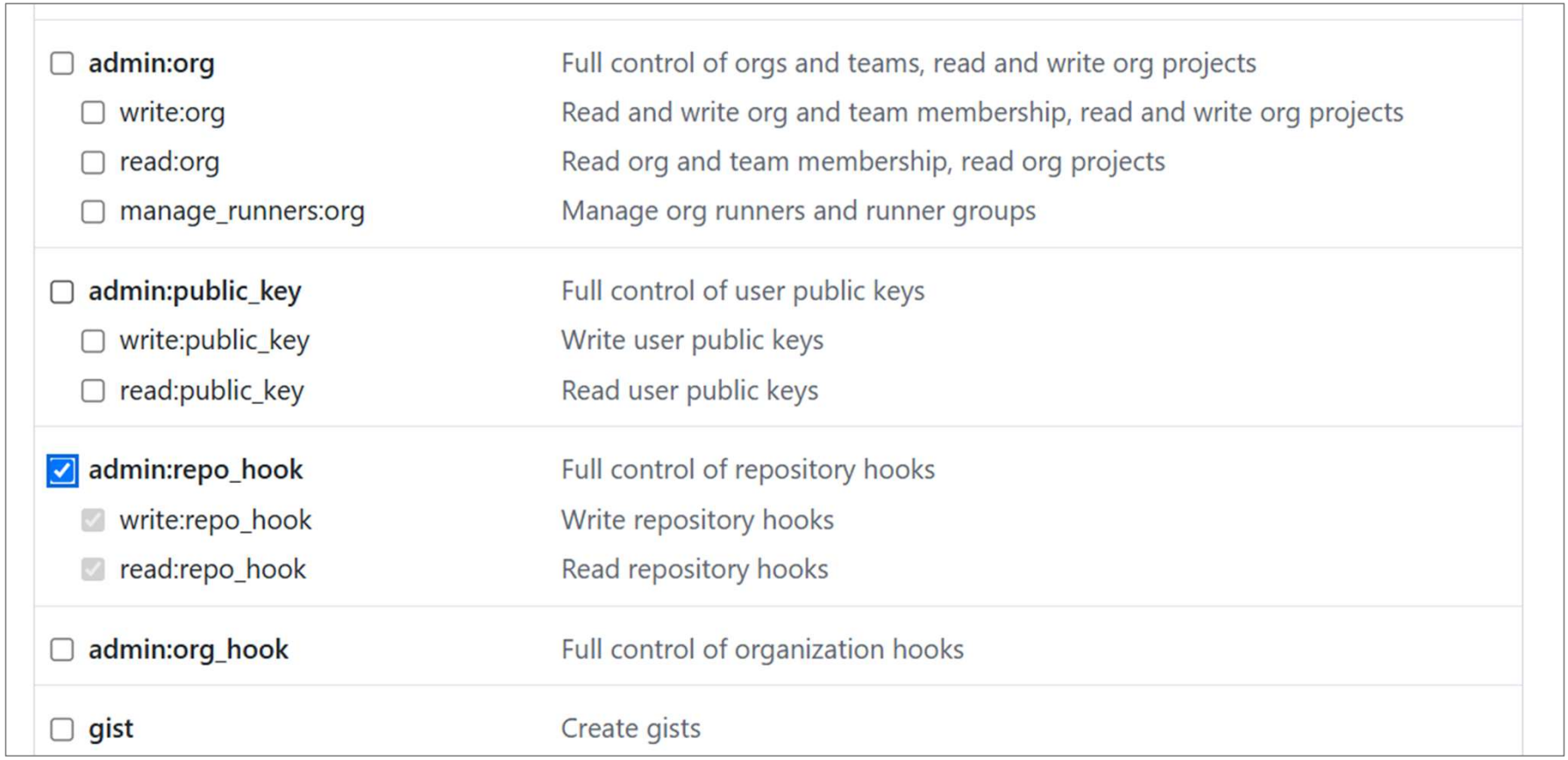




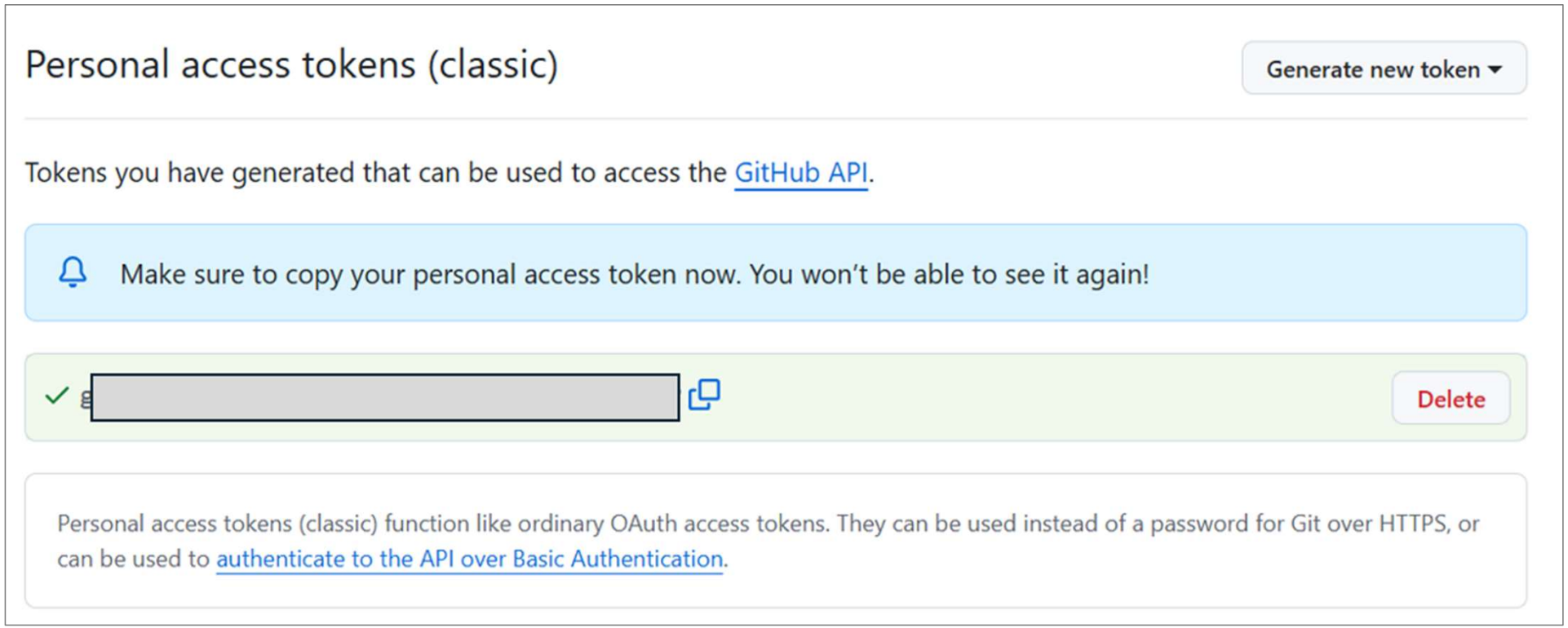
Credential 등록 – github access token 발급 (계속)



Credential 등록 – github access token 발급 (계속)

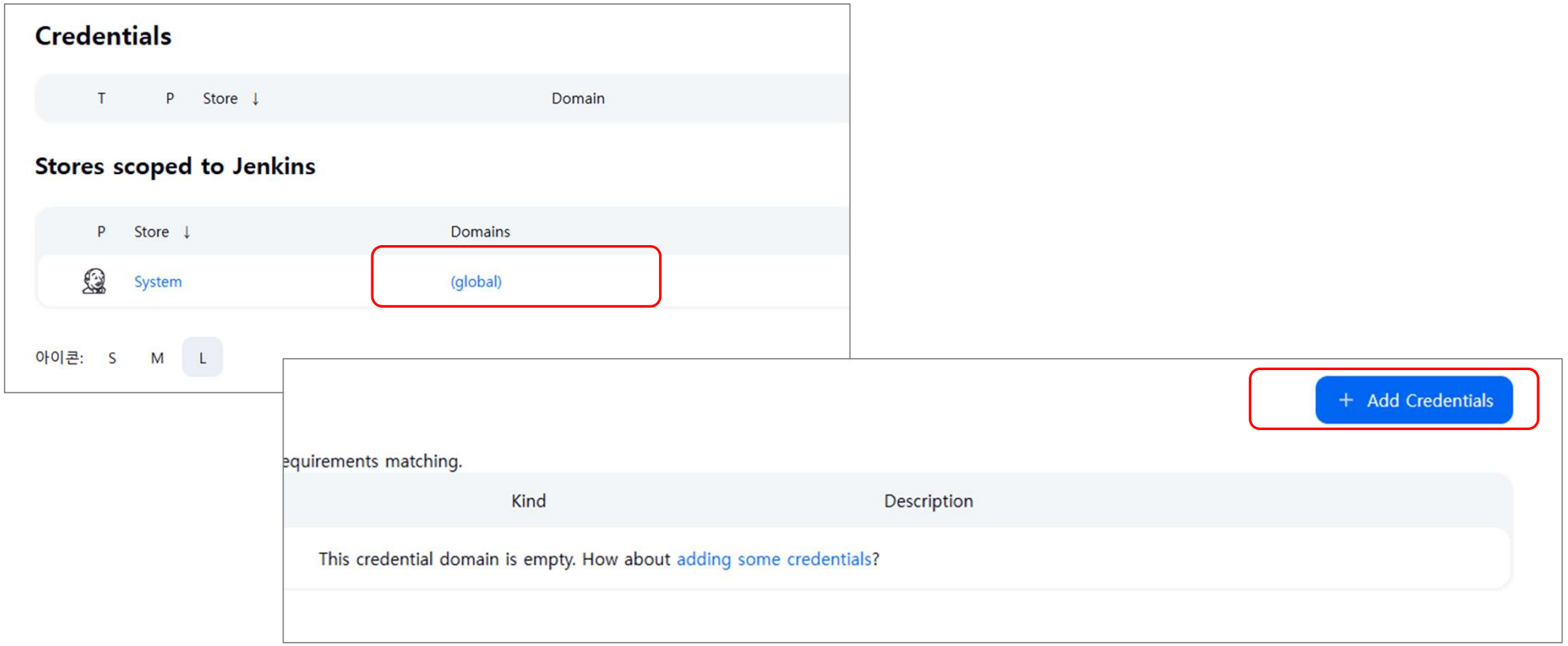


Credential 등록 – github access token 발급 (계속)

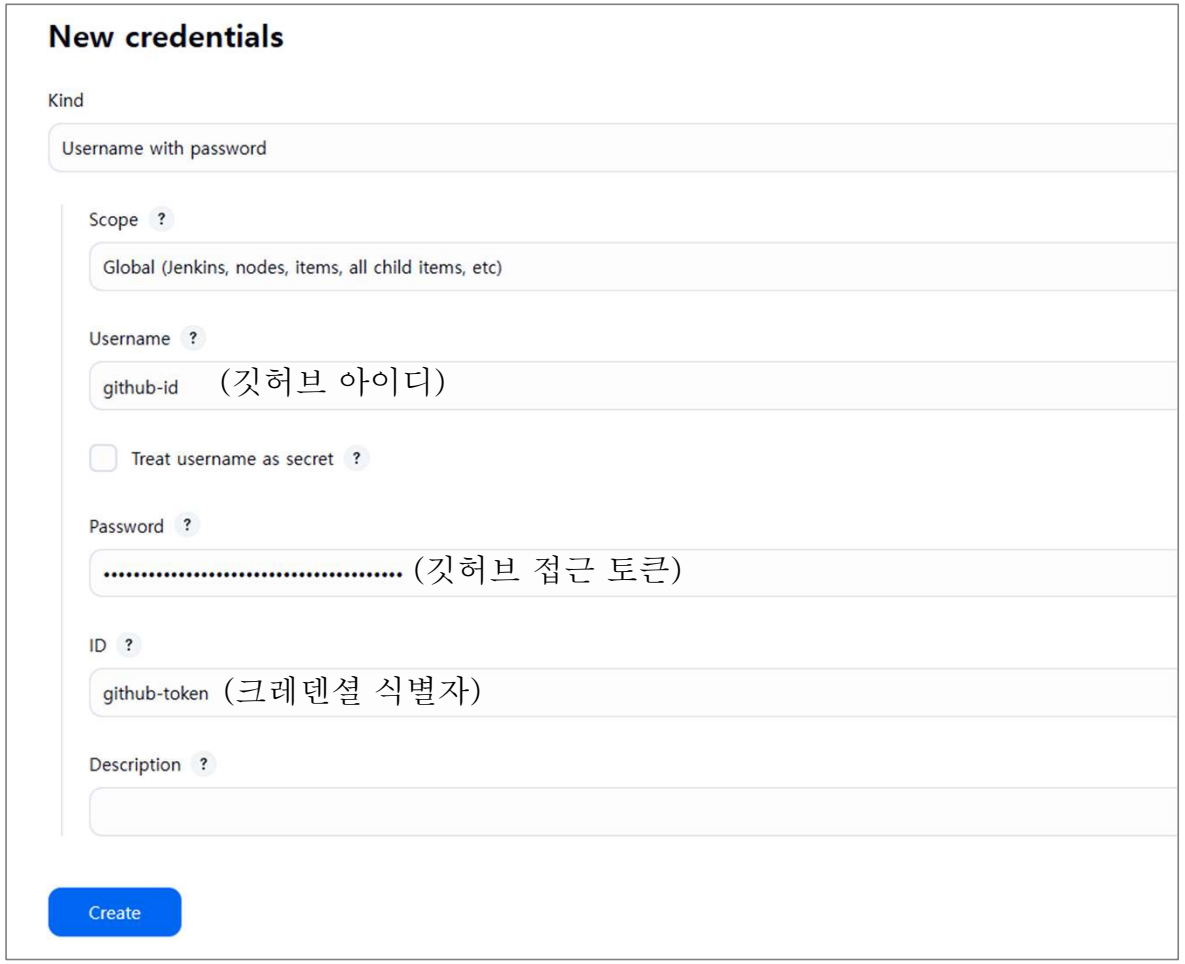


Credential 등록 – github access token 등록

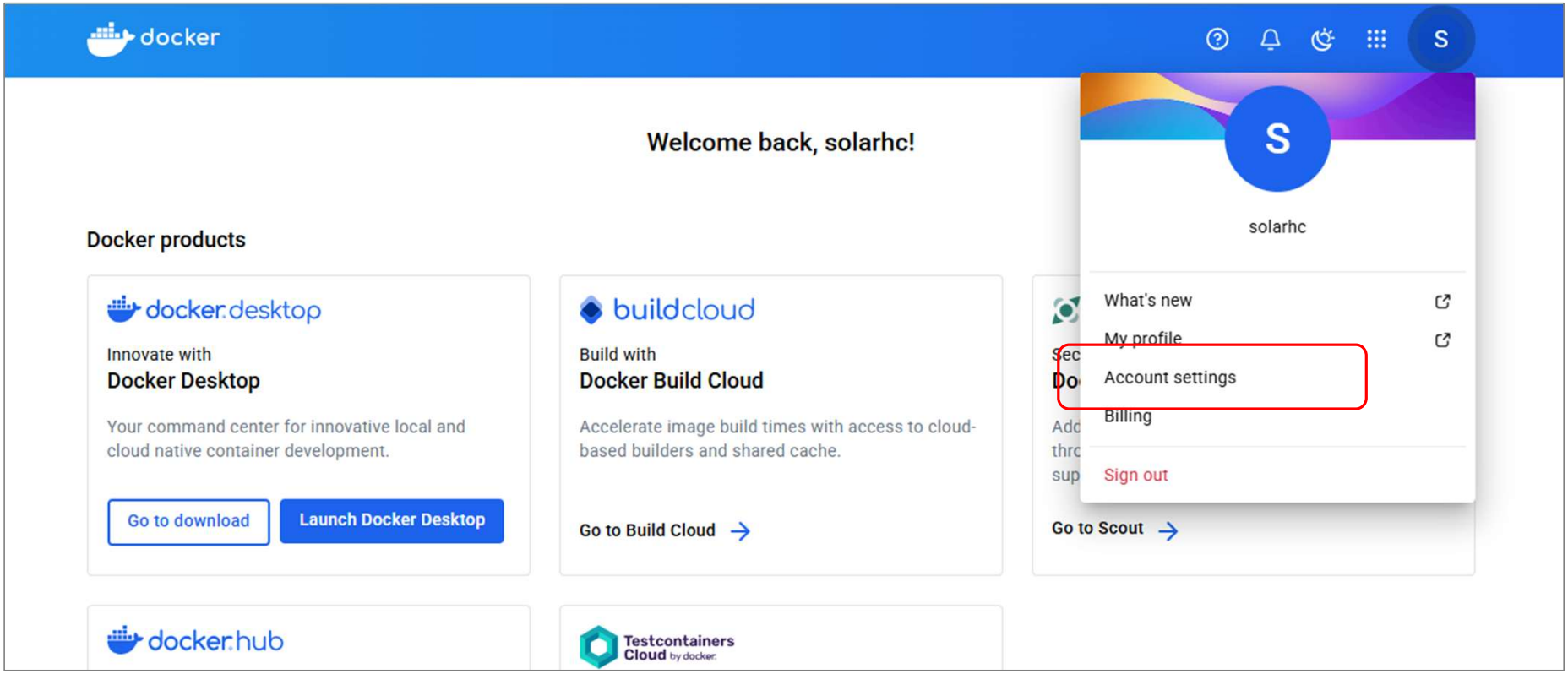
Jenkins 관리 > Credentials



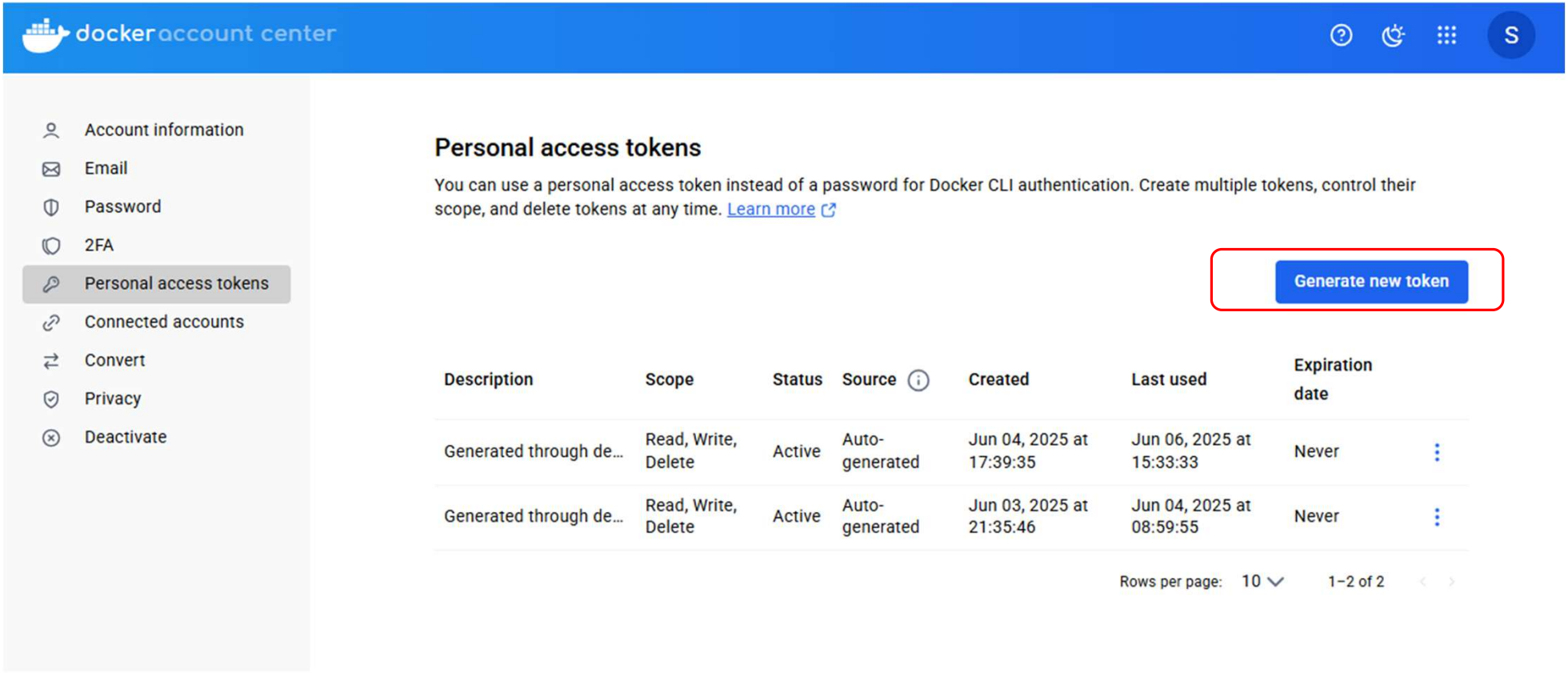
Credential 등록 – github access token 등록 (계속)



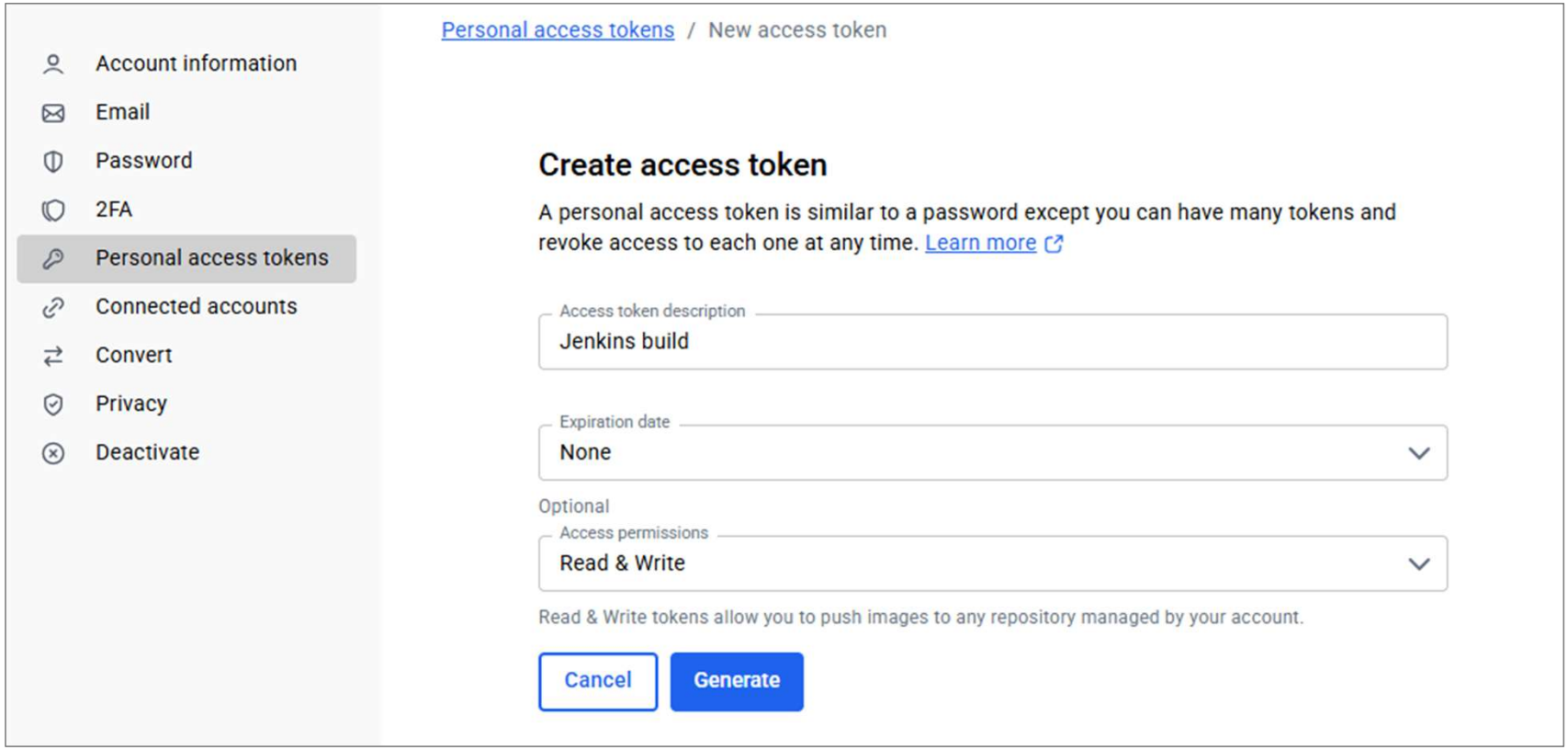
Credential 등록 – dockerhub access token 발급



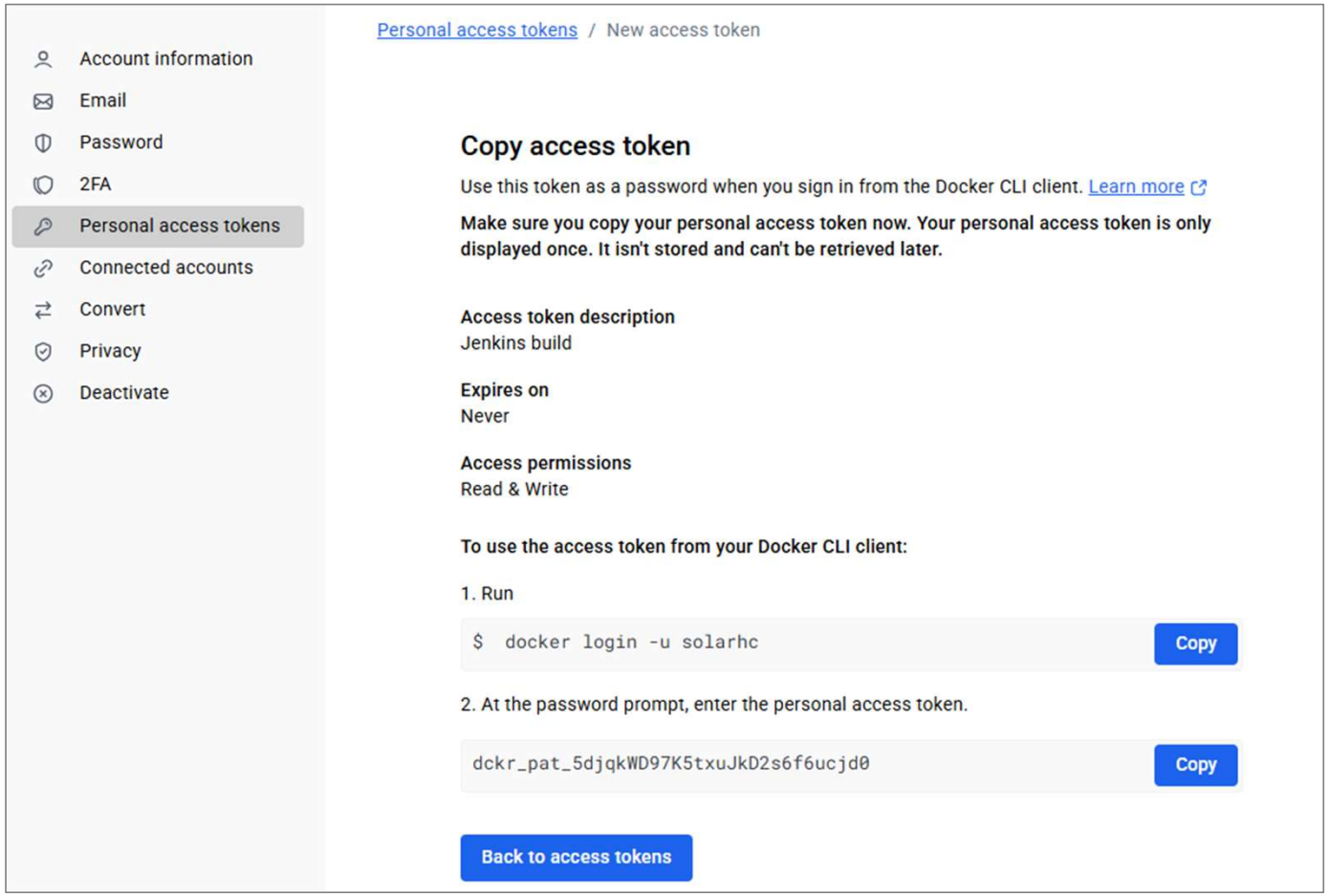
Credential 등록 – dockerhub access token 발급 (계속)



Credential 등록 – dockerhub access token 발급 (계속)

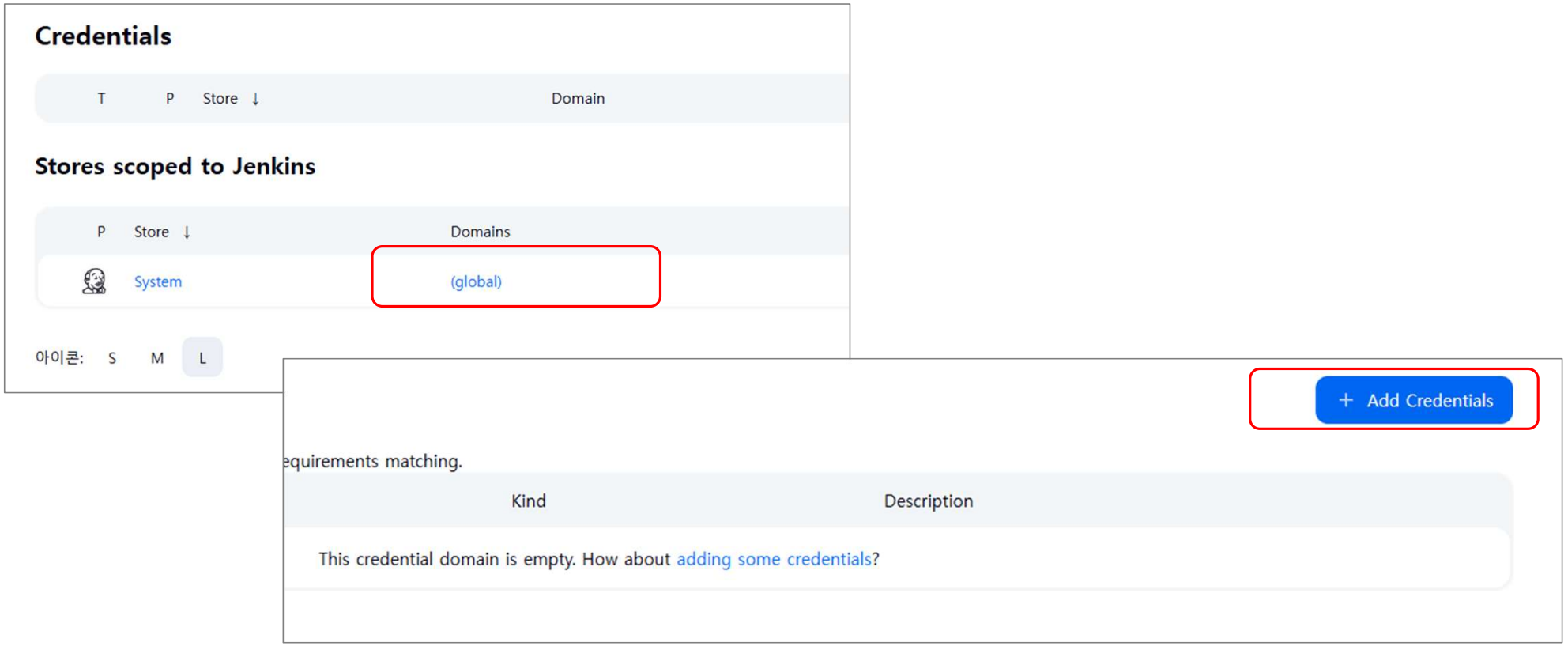


Credential 등록 – dockerhub access token 발급 (계속)

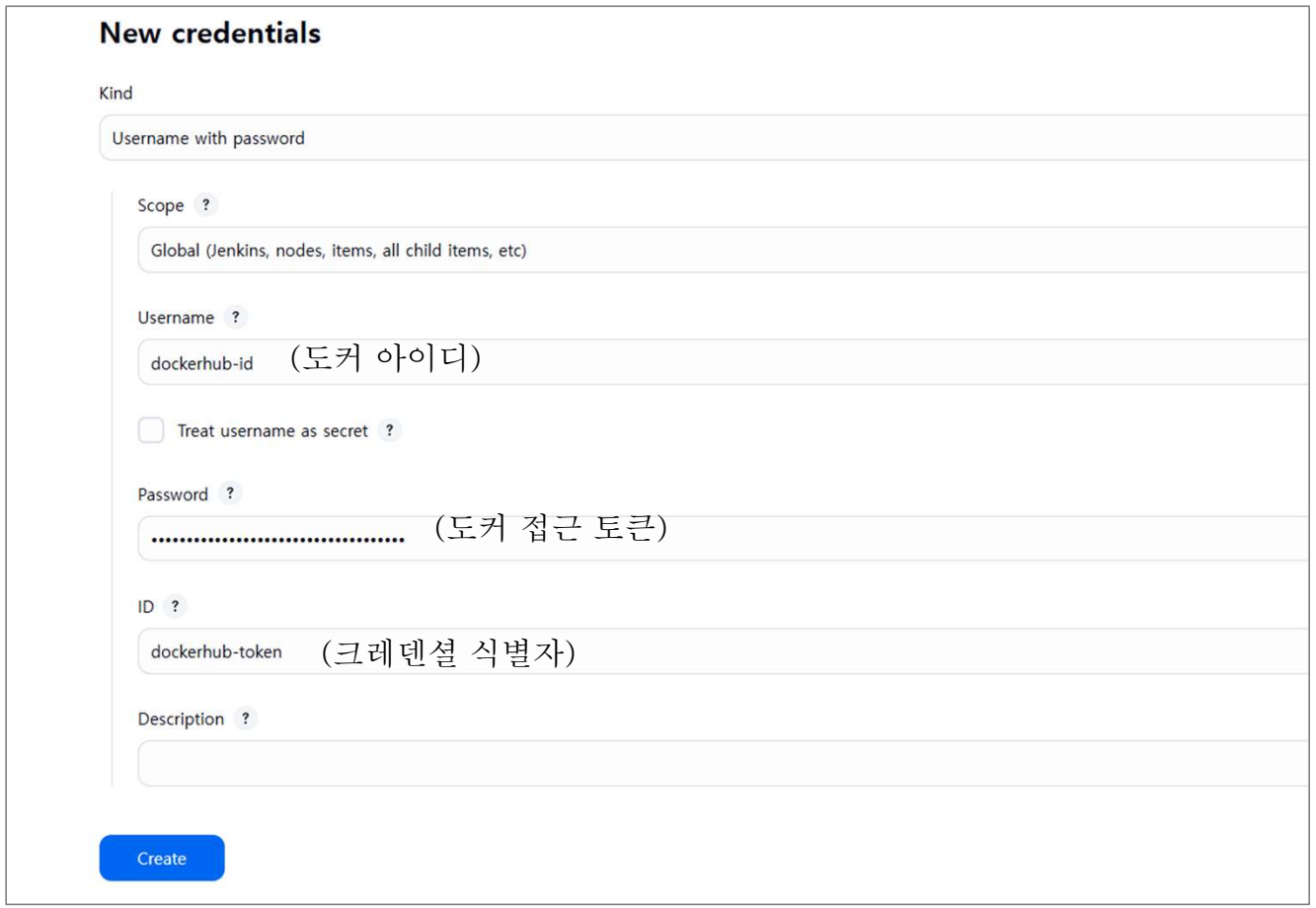


Credential 등록 – dockerhub access token 등록

Jenkins 관리 > Credentials

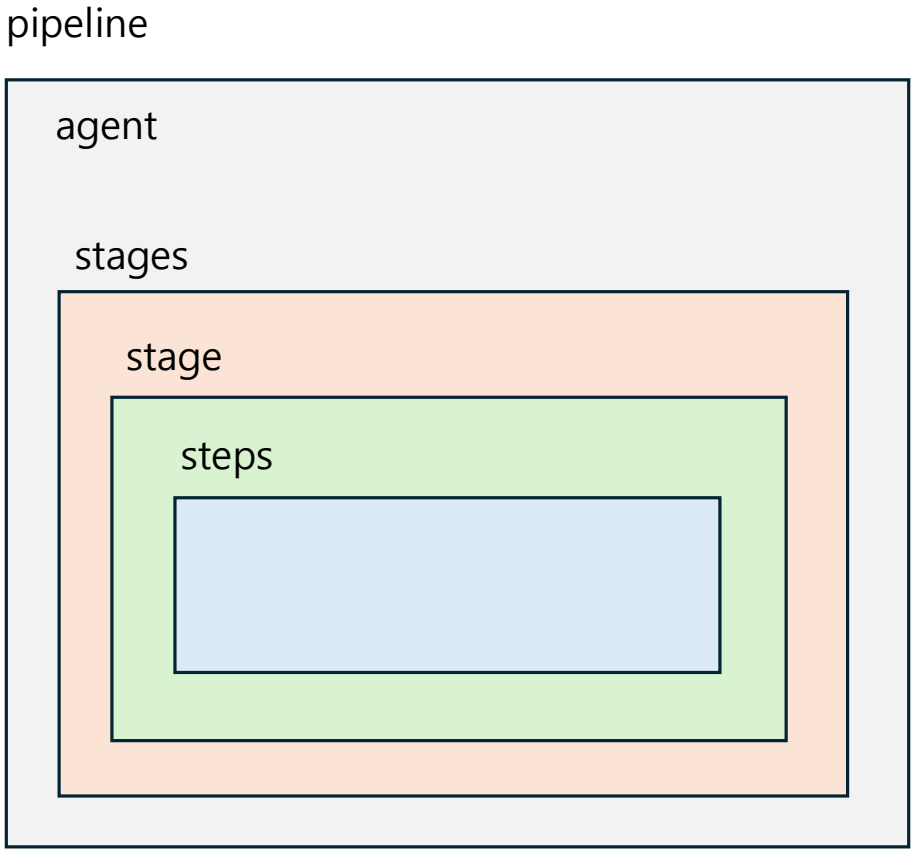


Credential 등록 – dockerhub access token 등록 (계속)



Pipeline

파이프라인 기본 구조




```
pipeline {
    agent any

    stages {

        stage('stage name') {

            steps {
                ...
            }
        }
    }
}
```

파이프라인 문법 – pipeline

```
pipeline {
    /* insert Declarative Pipeline here */
}
```

- 파이프라인을 정의하기 위해서는 반드시 pipeline을 포함해야 한다.
- pipeline은 최상위 레벨이 되어야하며, { }으로 정의해야 한다.
- pipeline은 Section, Directives, Steps 또는 assignment 문으로 구성되어야 한다.

- Section : agent, post, stages, steps
- Directives: environment, options, parameters, triggers, tools

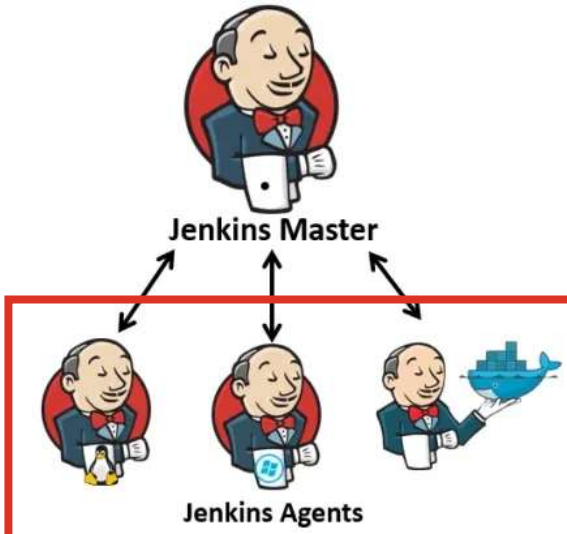
파이프라인 문법 – (section) agent

```
pipeline {
    agent {
        node {
            label 'master'
        }
    }

    …
```

master인 노드에서 실행

- Jenkins 환경에서 전체 Pipeline이나 특정 stage가 실행될 위치를 지정
- pipeline { } 블록 내의 최상단에 정의
- 각 stage { } 블록의 최상단에도 정의가 가능하지만, 필수 사항은 아님

- agent any: 사용 가능한 agent.
- agent none: global agent는 설정 x. 각 stage에 설정 필요
- agent { node { ... } }



Jenkins Agent
- 실제로 빌드, 테스트, 배포 작업 수행

파이프라인 문법 – (section) stages

```
pipeline {
    agent any

    stages {
        stage('Build') {
            ...
        }

        stage('Test') {
            ...
        }
    }
}
```

- 하나 이상의 stage를 포함하는 컨테이너
- Pipeline이 말하는 work의 대부분이 위치하는 곳

파이프라인 문법 – (section) steps

```
pipeline {
    agent any

    stages {
        stage('Build & Test Application') {
            steps {
                sh "gradle clean build"
            }
        }
    }
}
```

- stage 내부 실제 실행 명령을 정의

파이프라인 문법 – (section) post

```
pipeline {
    agent any

    stages {
        stage('Build & Test Application') {
            steps {
                sh "gradle clean build"
            }
        }
    }

    post {
        success { slackSend channel: '#deploy', message: 'Success!' }
        failure { mail to: 'team@example.com', subject: 'Pipeline Failed!' }
    }
}
```

- 파이프라인/스테이지 실행 후 조건부 작업을 정의

파이프라인 문법 – (directives) environment

```
pipeline {
    agent {
        node {
            label 'master'
        }
    }

    environment {
        GIT_URL = "https://github.com/solarhc/k8s-backend-user.git"
        GITHUB_CREDENTIAL = "github-token"
        DOCKERHUB_CREDENTIAL = 'dockerhub-token'
    }

    …
```

- key-value 형태로 파이프라인 내부에서 사용할 환경 변수로 선언

파이프라인 문법 – (directives) parameters

```
pipeline {
    agent {
        node {
            label 'master'
        }
    }

    parameters {
        booleanParam defaultValue: false, description: '', name: 'RELEASE'
    }

    …
```

- 사용자 입력 매개변수를 정의

파이프라인 문법 – (directives) options

```
pipeline {
    …

    options {
        disableConcurrentBuilds()
        buildDiscarder(logRotator(numToKeepStr: "30", artifactNumToKeepStr: "30"))
        timeout(time: 120, unit: 'MINUTES')
    }

    …
```

- 파이프라인 전역 옵션을 설정

파이프라인 문법 – (directives) tools

```
pipeline {
    agent {
        node {
            label 'master'
        }
    }
    …
    tools {
        gradle 'Gradle 8.14.2'
        jdk 'OpenJDK 17'
        dockerTool 'Docker'
    }
}
```

미리 정의된 도구의 아이디

- 파이프라인에서 빌드 도구의 자동 설치 및 환경 구성을 지원하는 지시문
- 주로 JDK, Maven, Gradle 등 빌드에 필요한 도구를 지정
- Jenkins의 Global Tool Configuration에서 미리 정의된 도구만 사용 가능

파이프라인 문법 – (directives) when

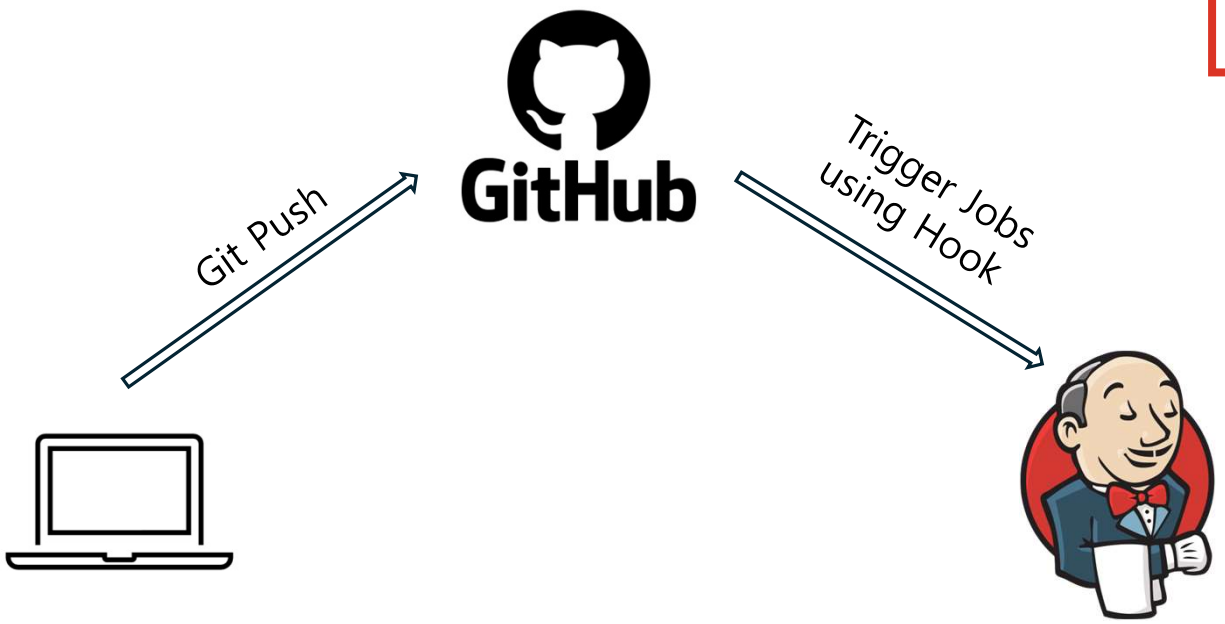
실행 환경이 'prod'인 경우, docker image를 push 하도록 실행

```
pipeline {
    …
    stages {
        stage('Deliver to Prod Repository') {
            when {
                // 조건문 (예: 브랜치, 환경  변수, 파라미터  등)
                expression { params.ENV == 'prod' }
            }
            steps {
                script {
                    docker.withRegistry("", DOCKERHUB_CREDENTIAL) {
                        docker.image("${DOCKER_IMAGE_NAME}").push()
                    }
                }
            }
        }
        …
    }
```

- 특정 조건이 충족될 때만 Stage를 실행하도록 제어하는 지시문
- 조건부 빌드, 환경별 배포, 브랜치/변수 기반 분기 처리에 유용

빌드 트리거

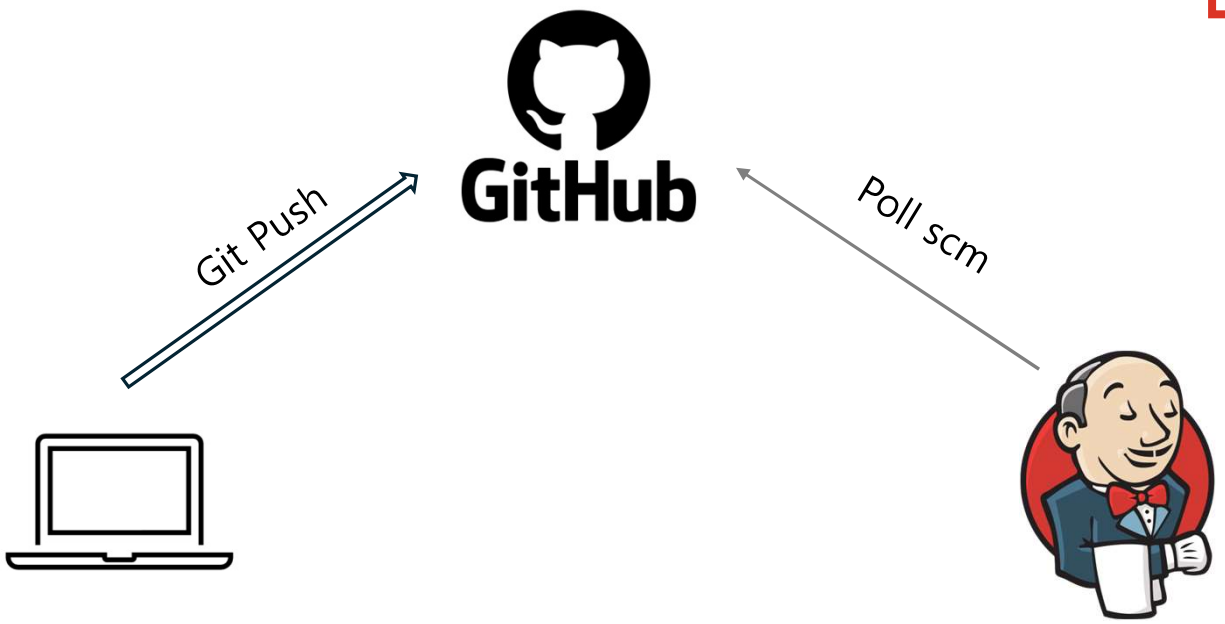
빌드 트리거 – Push Notification 방식



Jenkins 트리거
- Generic Webhook Trigger
- Github hook trigger for GITScm polling

주의) Jenkins 서버가 Github에서 접근 가능해야 함 (공개 IP)

빌드 트리거 – Polling 방식



Jenkins 트리거
- Poll SCM

- 주기적으로 소스 코드 저장소를 폴링하여 변경사항을 감지하고, 변경이 있을 경우 빌드 트리거 하는 방식
- 크론(Cron) 방식으로 스케쥴 설정
- 웹훅 설정이 불가능한 환경에서 사용

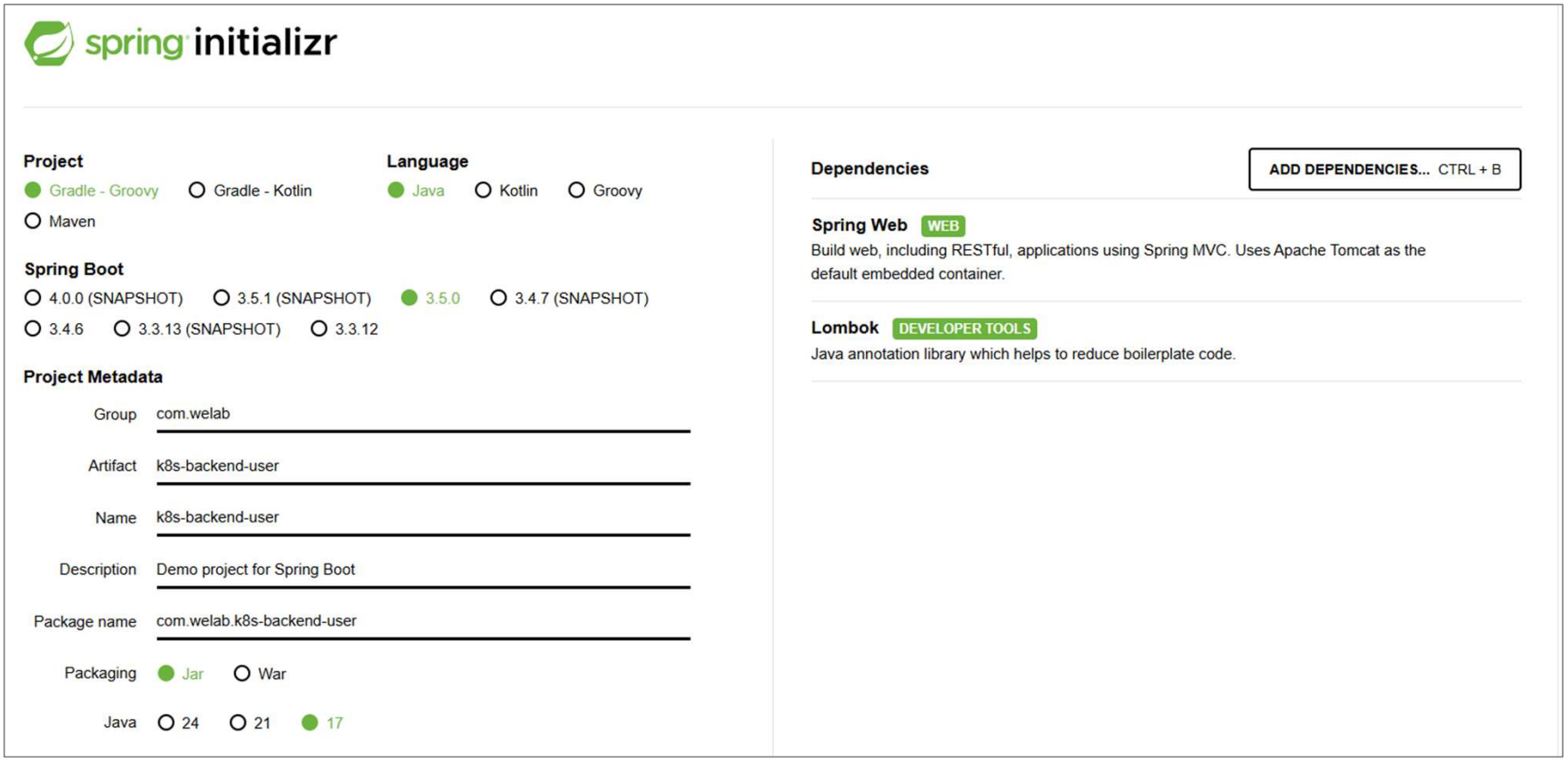
분 시 일 월 요일
(스프링 스케줄러와 동일한 방식)

실습

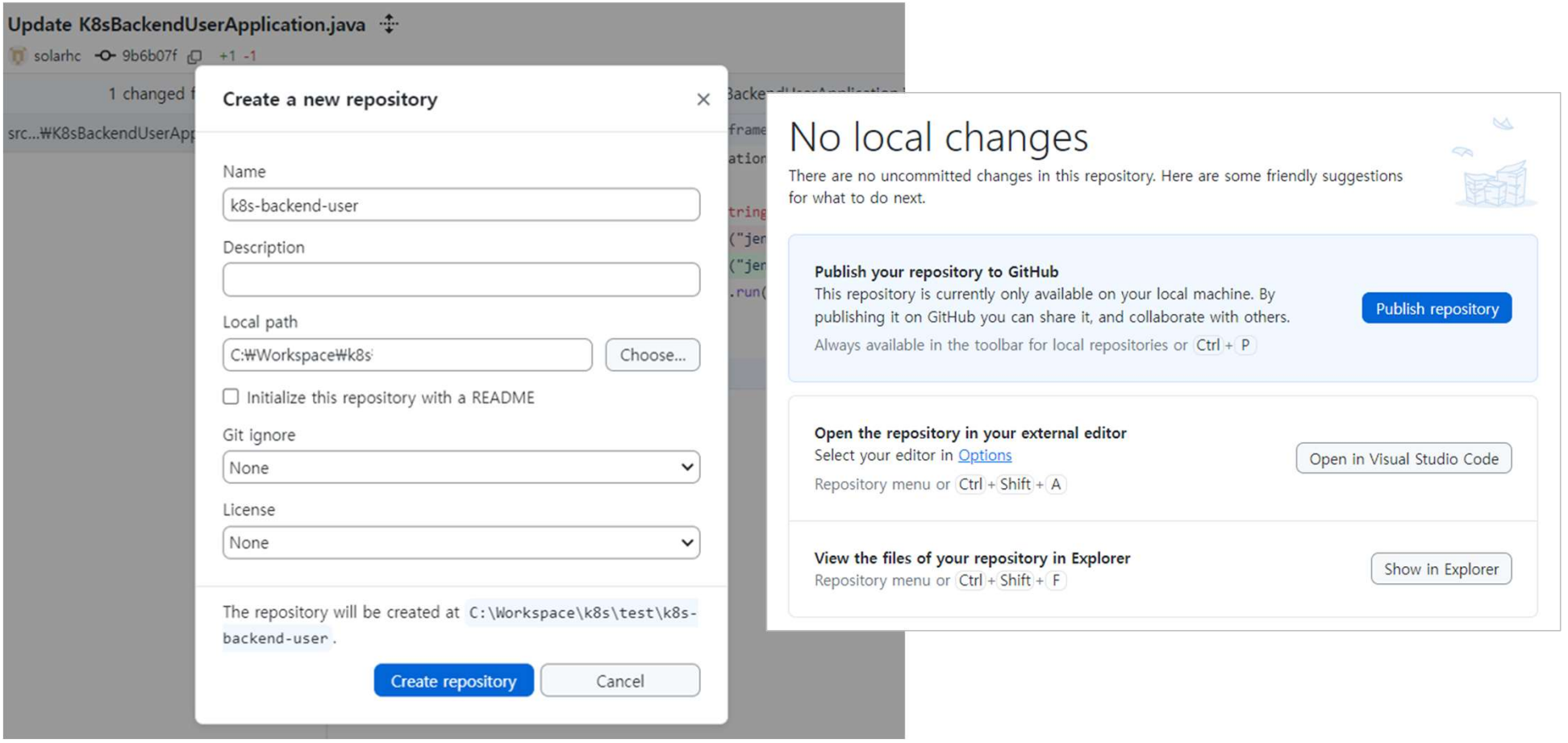
프로젝트 생성 – User project

https://start.spring.io/ 접속

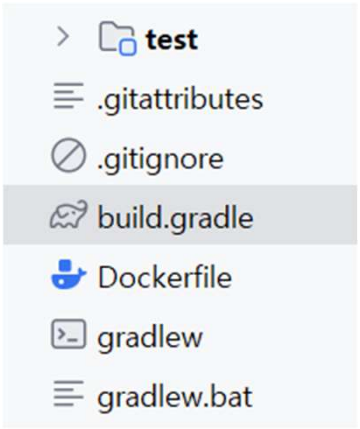
Artifact: k8s-backend-user



Create & Publish Repository



User 프로젝트 - build.gradle 수정

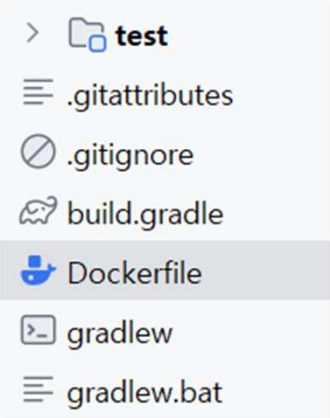


settings.gradle 내에 설정된 rootProject name을 출력

프로젝트의 버전 출력

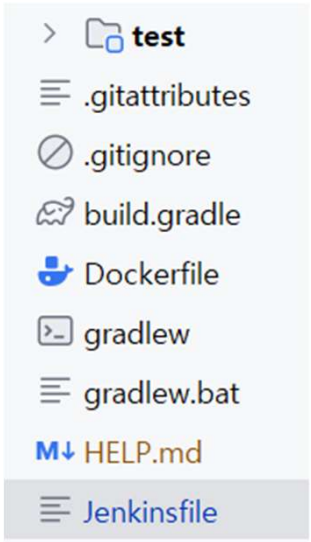
```
tasks.named('test') {
    useJUnitPlatform()
}

jar {
    enabled = false // plain.jar 생성 완전히 비활성화
}

tasks.register('getAppName') {
    doLast {
        println "${rootProject.name}"
    }
}

tasks.register('getAppVersion') {
    doLast {
        println "${project.version}"
    }
}
```

User 프로젝트 - Dockerfile 작성



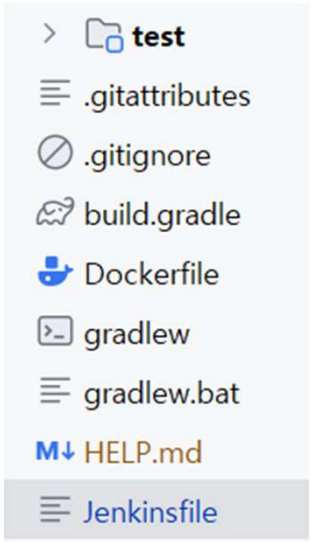

```
FROM amazoncorretto:17
MAINTAINER dev@welab.com
VOLUME /tmp
EXPOSE 8080
COPY build/libs/*.jar /app.jar
ENTRYPOINT ["java", "-Djava.security.egd=file:/dev/./urandom", "-jar", "/app.jar"]
```

User 프로젝트 - Jenkinsfile 작성




```
def APP_NAME
def APP_VERSION
def DOCKER_IMAGE_NAME
def PROD_BUILD = false
pipeline {
    agent {
        node {
            label 'master'
        }
    }

    parameters {
        gitParameter branch: '',
                     branchFilter: '.*',
                     defaultValue: 'origin/main',
                     description: '', listSize: '0',
                     name: 'TAG',
                     quickFilterEnabled: false,
                     selectedValue: 'DEFAULT',
                     sortMode: 'DESCENDING_SMART',
                     tagFilter: '*',
                     type: 'PT_BRANCH_TAG'

        booleanParam defaultValue: false, description: '', name: 'RELEASE'
    }
```

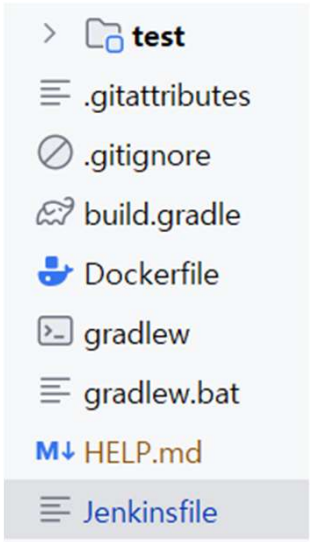
User 프로젝트 - Jenkinsfile 작성 (계속)



[redacted] : 자신의 것으로 수정 必

```
environment {
    GIT_URL = "https://github.com/[redacted]/k8s-backend-user.git"
    GITHUB_CREDENTIAL = "github-token"
    ARTIFACTS = "build/libs/**"
    DOCKER_REGISTRY = "[redacted]"
    DOCKERHUB_CREDENTIAL = 'dockerhub-token'
}

options {
    disableConcurrentBuilds()
    buildDiscarder(logRotator(numToKeepStr: "30", artifactNumToKeepStr: "30"))
    timeout(time: 120, unit: 'MINUTES')
}

tools {
    gradle 'Gradle 8.14.2'
    jdk 'OpenJDK 17'
    dockerTool 'Docker'
}
```

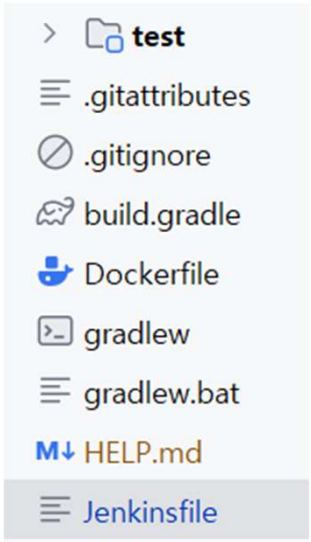
User 프로젝트 - Jenkinsfile 작성 (계속)




```
stages {
    stage('Set Version') {
        steps {
            script {
                APP_NAME = sh (
                        script: "gradle -q getAppName",
                        returnStdout: true
                ).trim()
                APP_VERSION = sh (
                        script: "gradle -q getAppVersion",
                        returnStdout: true
                ).trim()

                DOCKER_IMAGE_NAME = "${DOCKER_REGISTRY}/${APP_NAME}:${APP_VERSION}"

                sh "echo IMAGE_NAME is ${APP_NAME}"
                sh "echo IMAGE_VERSION is ${APP_VERSION}"
                sh "echo DOCKER_IMAGE_NAME is ${DOCKER_IMAGE_NAME}"

            }
        }
    }
```

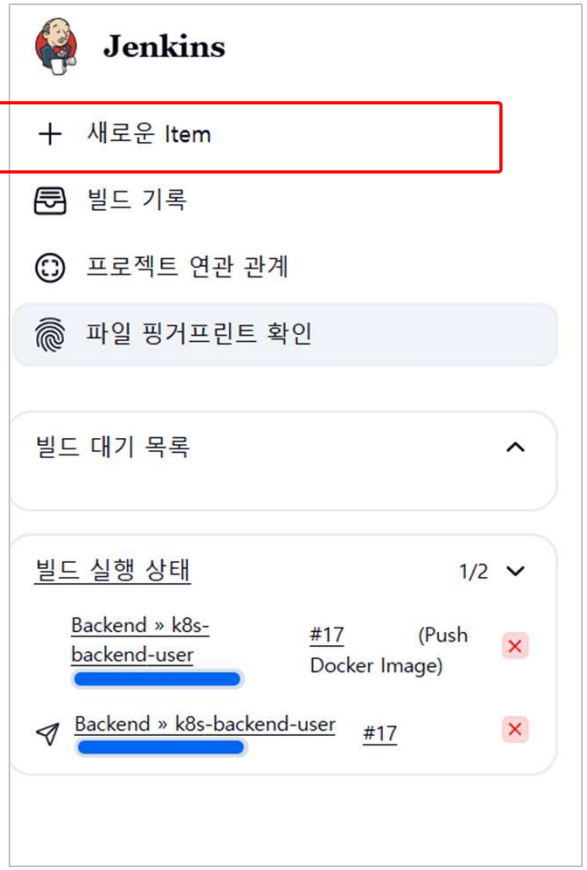
build.gradle에서 정의한 명령
(출력 형식으로 반환됨)

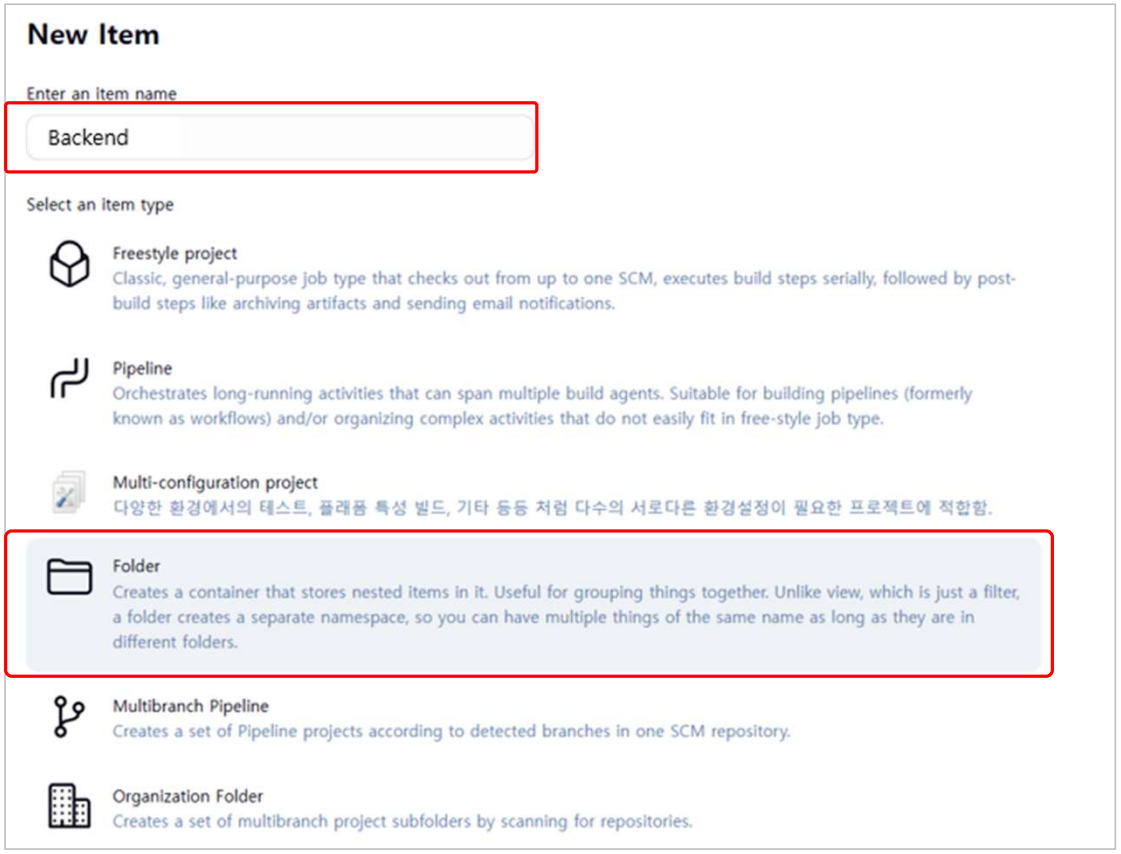
User 프로젝트 - Jenkinsfile 작성 (계속)



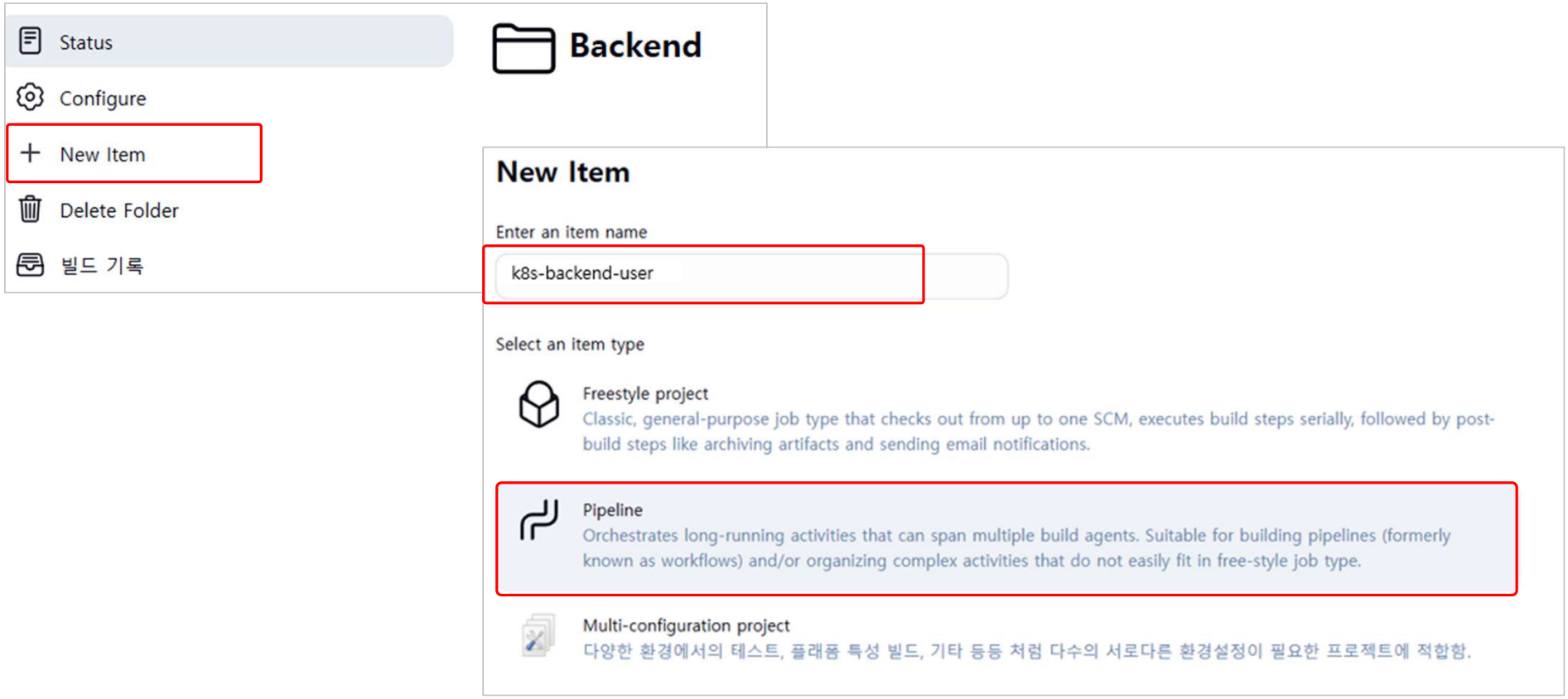

```
        stage('Build & Test Application') {
            steps {
                sh "gradle clean build"
            }
        }

        stage('Build Docker Image') {
            steps {
                script {
                    docker.build "${DOCKER_IMAGE_NAME}"
                }
            }
        }

        stage('Push Docker Image') {
            steps {
                script {
                    docker.withRegistry("", DOCKERHUB_CREDENTIAL) {
                        docker.image("${DOCKER_IMAGE_NAME}").push()
                    }
                    sh "docker rmi ${DOCKER_IMAGE_NAME}"
                }
            }
        }
    }
}
```

Jenkins – Backend Folder 생성

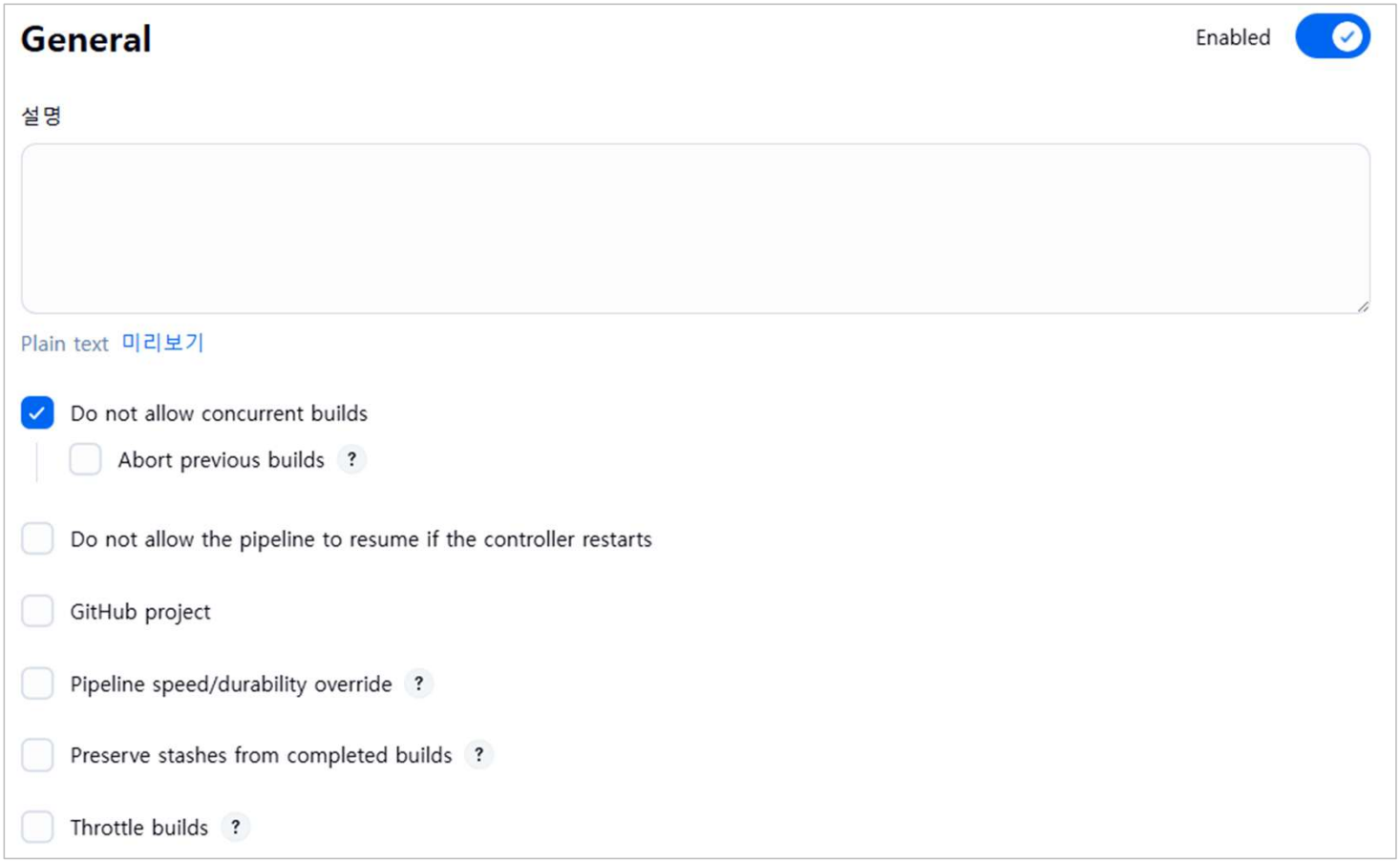




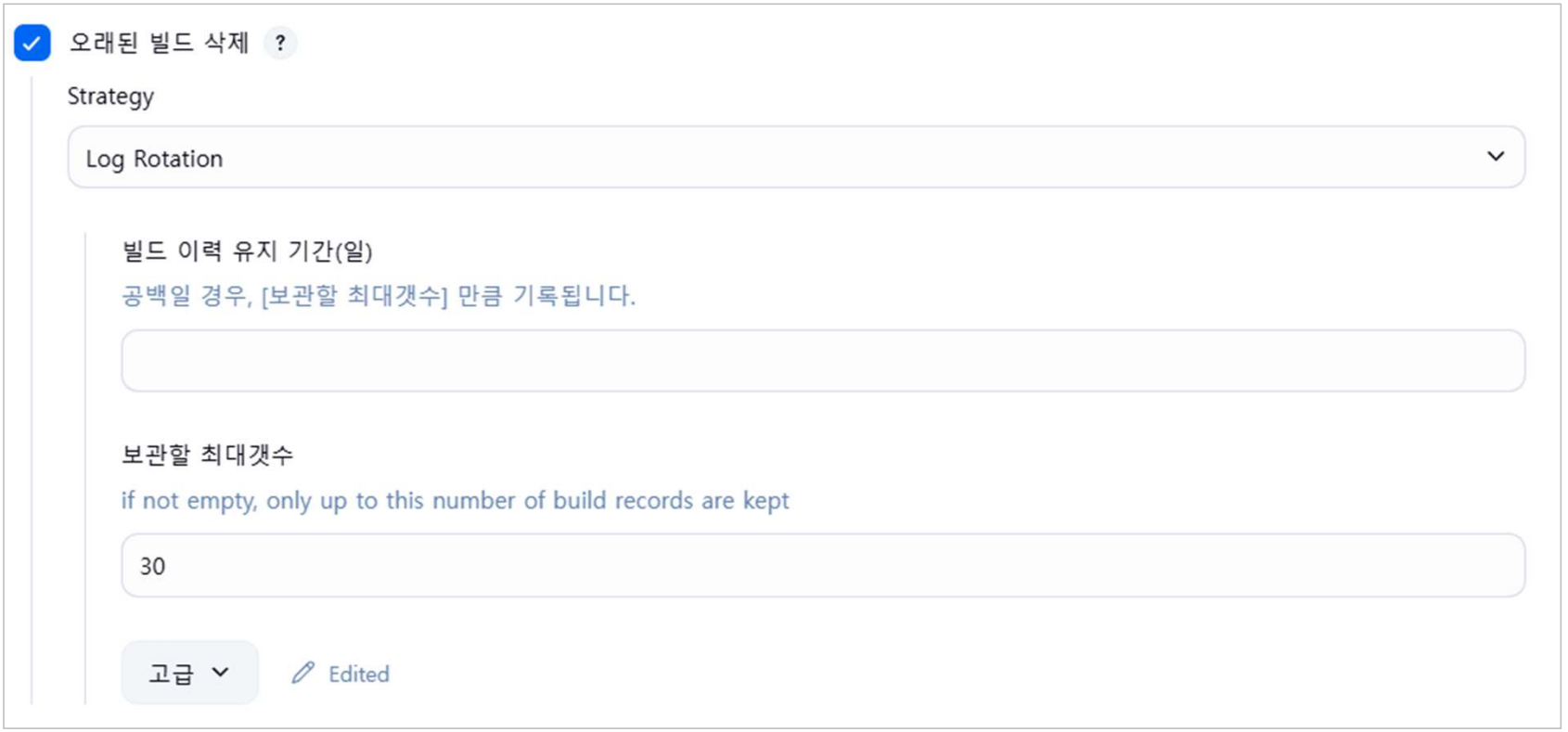
Jenkins – k8s-backend-user 파이프라인 생성



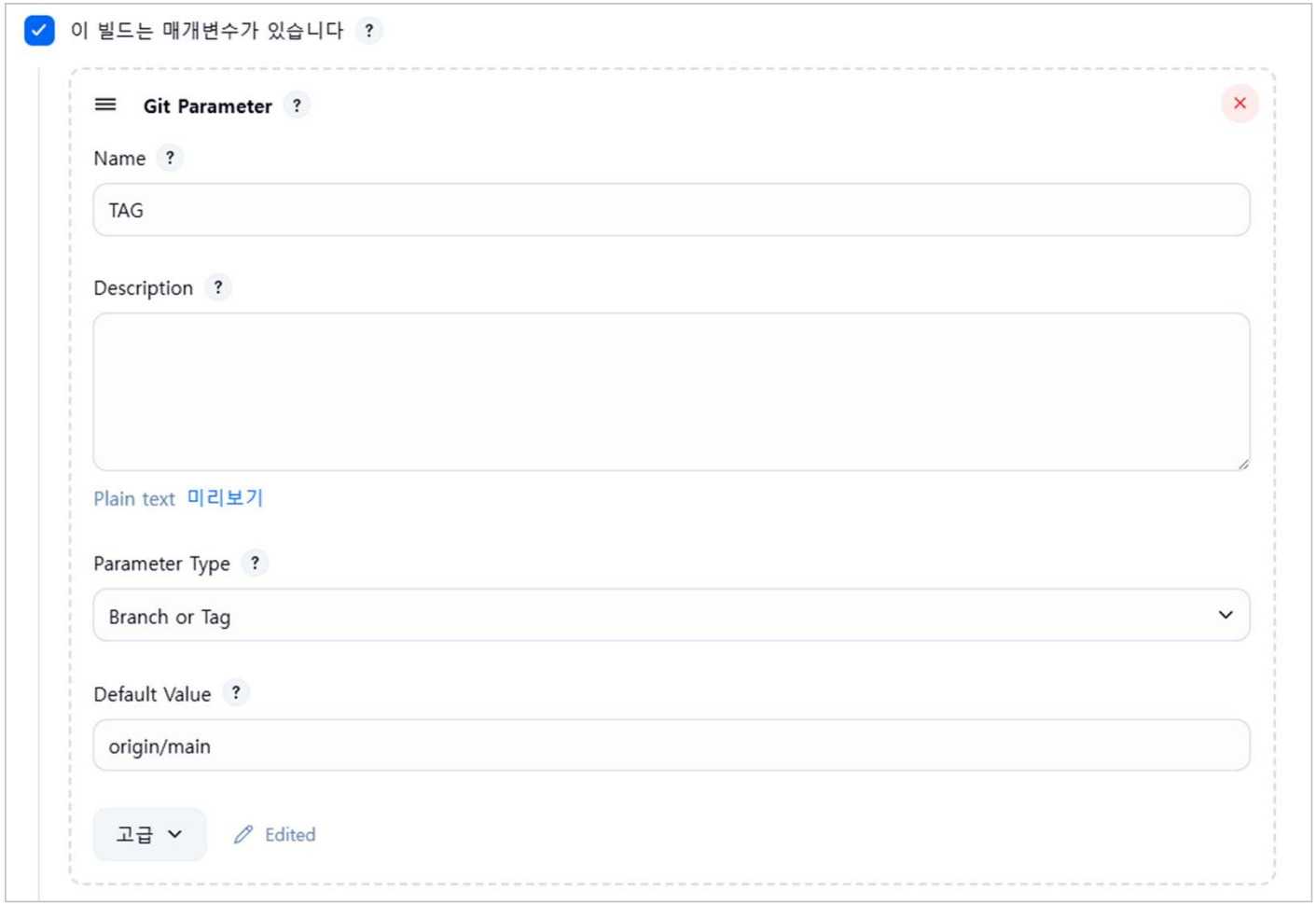
Jenkins – k8s-backend-user 파이프라인 설정



Jenkins – k8s-backend-user 파이프라인 설정 (계속)



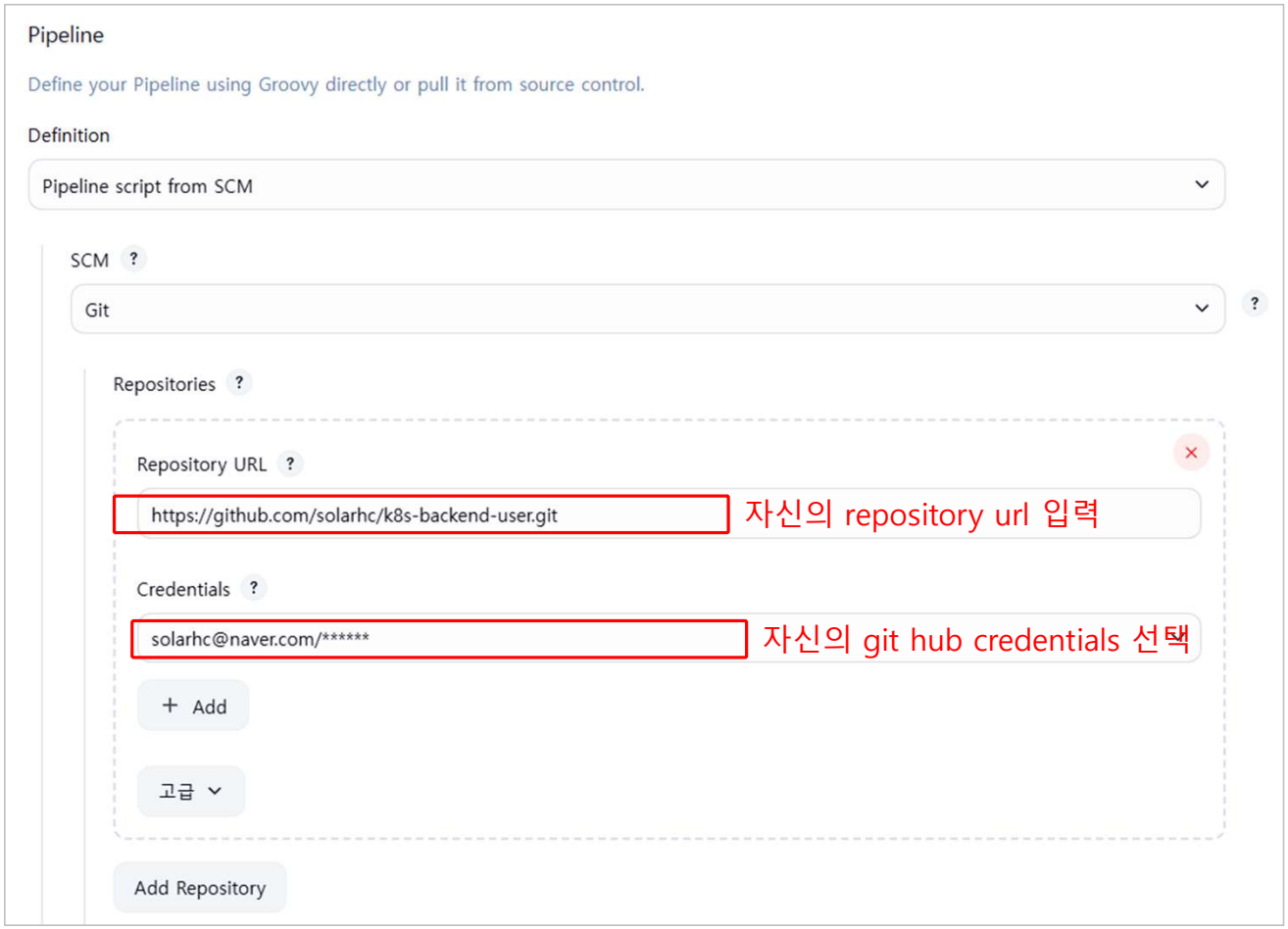
Jenkins – k8s-backend-user 파이프라인 설정 (계속)



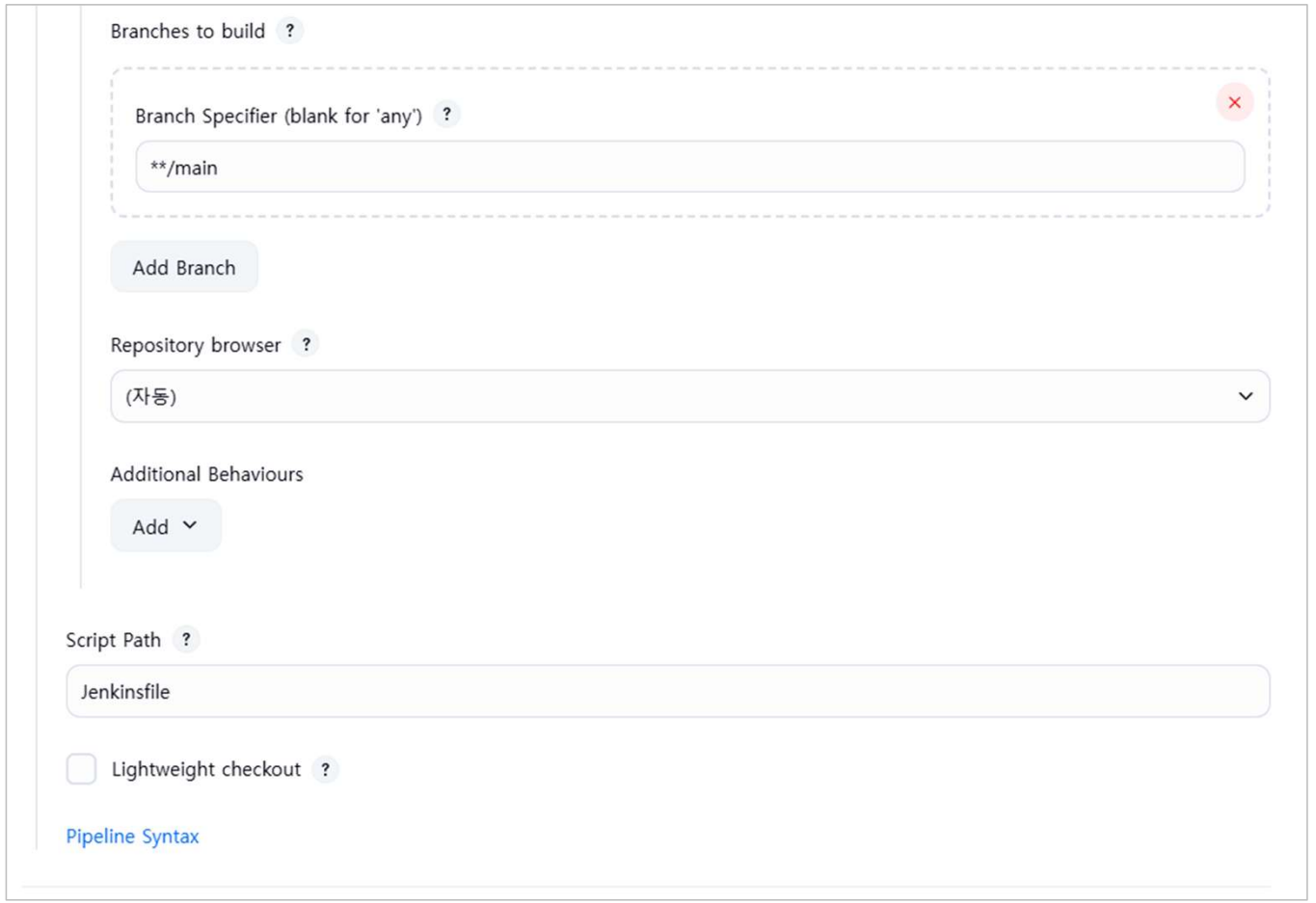
Jenkins – k8s-backend-user 파이프라인 설정 (계속)

Jenkins – k8s-backend-user 파이프라인 설정 (계속)

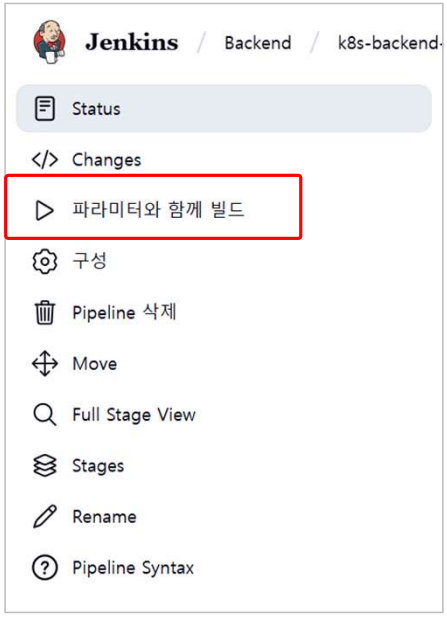
Jenkins – k8s-backend-user 파이프라인 설정 (계속)

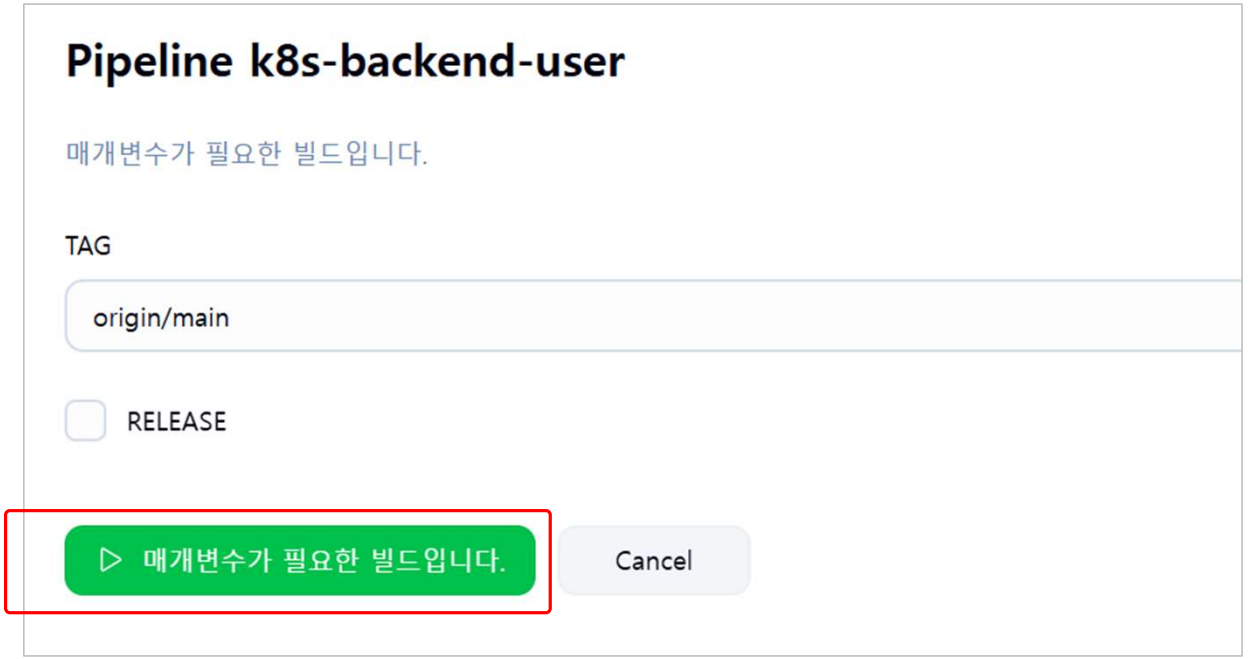


Jenkins – k8s-backend-user 파이프라인 설정 (계속)



Jenkins 파이프라인 빌드





조건부 Stage 실행

Jenkinsfile

```
...
def PROD_BUILD = false
def TAG_BUILD = false
pipeline {
    ...
    stages {
        stage('Set Version') {
            steps {
                script {
                    ...
                    if( params.TAG.startsWith('origin') == false && params.TAG.endsWith('/main') == false ) {
                        if( params.RELEASE == true ) {
                            DOCKER_IMAGE_NAME += '-RELEASE'
                            PROD_BUILD = true
                        } else {
                            DOCKER_IMAGE_NAME += '-TAG'
                            TAG_BUILD = true
                        }
                    }
                }
            }
        }
        ...
        stage('Build Docker Image') {
            when {
                expression { PROD_BUILD == true || TAG_BUILD == true }
            }
            steps {
                script {
                    docker.build "${DOCKER_IMAGE_NAME}"
                }
            }
        }
        ...
```

Main branch가 아닌 TAG를 지정하여 빌드하는 경우
PROD_BUILD 또는 TAG_BUILD 가 true로 설정됨

PROD_BUILD 또는 TAG_BUILD 가 true 인 경우에만 docker build를 수행함

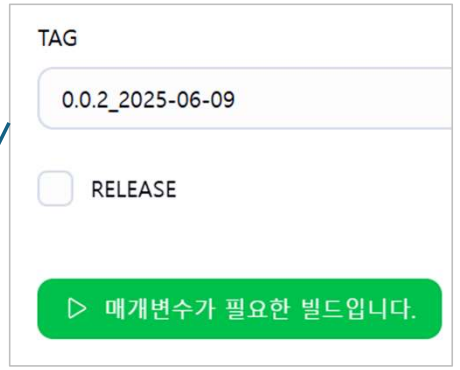
Jenkins Pipeline 구성 변경

위치: Jenkins Pipeline 구성 > Pipeline > SCM > Branches to build > Branch Specifier

Branch Specifier를 **/main에서 ${TAG} 로 변경

조건부 Stage 실행 결과

TAG 지정 빌드



main branch 빌드