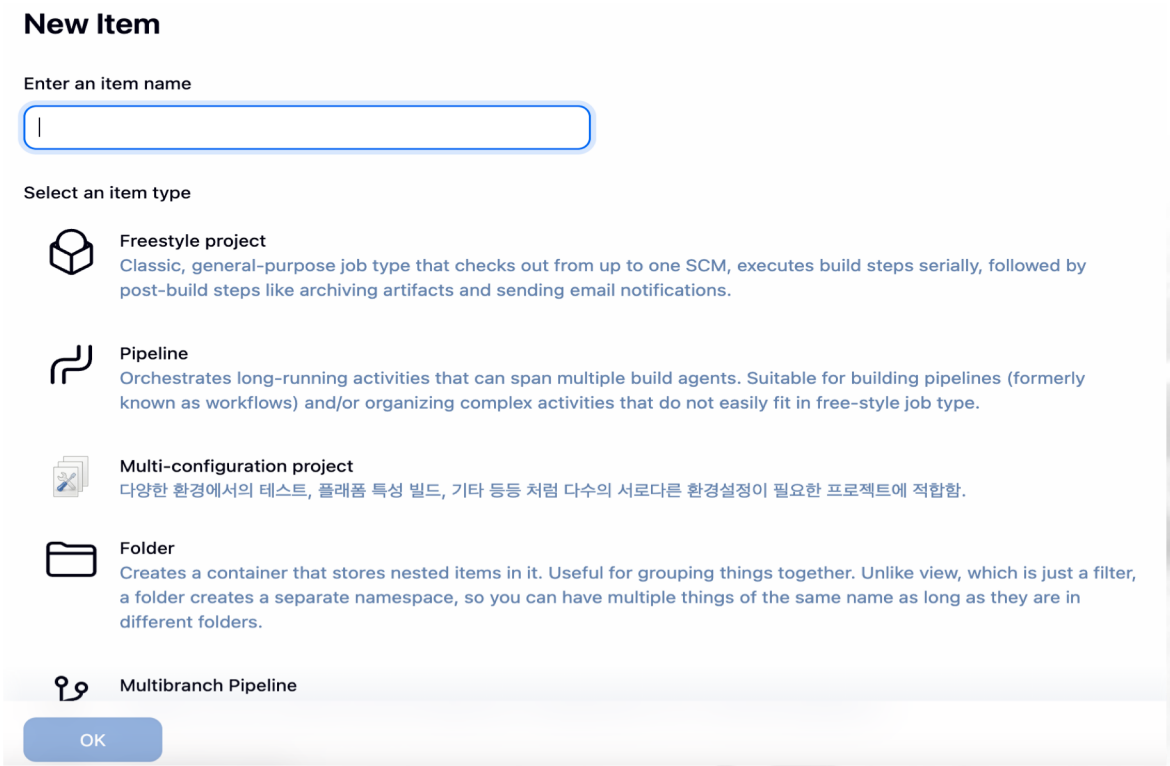
Jenkins FreeStyle

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - Jenkins에 로그인해 대시보드로 이동
    - 작업을 생성하기 위해 왼쪽 메뉴에서 New Item 링크를 클릭
    - 기존에 생성된 작업이 없는 경우에는 대시보드 화면 중간에 Create a Job 링크가 나타나는데 이 링크도 New Item 링크와 같은 역할을 수행



Free Style Job

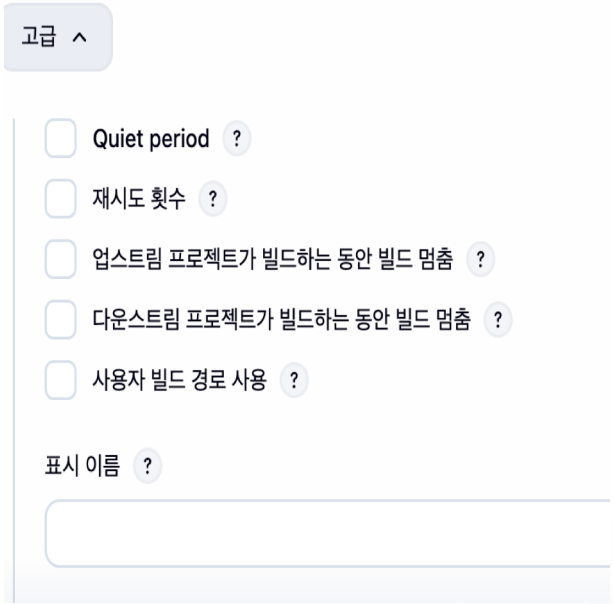
- Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - 작업 이름을 입력
    - Freestyle project 선택 후 OK

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ Description: 작업에 대한 설명이나 목적을 입력
      - ◆ GitHub project: GitHub Project를 설정하면 빌드 페이지에서 GitHub 저장소로 바로 이동할 수 있는 링크를 제공하고 Webhook 트리거와의 연계도 편리한데 GitHub Plugin 을 설치해야 사용 가능
      - ◆ Throttle builds: 동시에 실행될 빌드 수를 제한하여 서버 리소스 과부하를 방지하거나 특정 Job을 순차적으로 실행시키는 데 사용되는데 이 기능은 Throttle Concurrent Builds Plugin 설치 후 활성화할 수 있음
      - ◆ Discards old builds(오래된 빌드 삭제): 제한 값을 초과하면 오래된 빌드를 삭제하는 옵션
      - ◆ This project is parameterized(이 프로젝트는 매개변수가 있습니다.): 프로젝트를 빌드할 때 외부에서 매개변수를 대입받을 수 있는 프로젝트
      - ◆ Execute concurrent builds if necessary(필요한 경우 concurrent 빌드 실행): 빌드 프로세스의 여러 단계를 병렬로 시작해서 작업 시간을 줄여주는 옵션

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성



Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ Quiet Period(대기 시간)
        - ▪ Quiet period대기 시간 옵션을 선택하면 새로운 빌드가 즉시 시작되지 않음
        - ▪ 빌드 큐에 추가되고 지정된 시간이 지나야 빌드 시작
        - ▪ 여러 개의 코드 커밋을 모아서 수행할 때 유용한 기능
        - ▪ 옵션을 선택하지 않은 상태에서 여러 개발자가 코드 리포지터리로 거의 동시에 커밋을 했다면 첫 번째 커밋이 이뤄지는 즉시 빌드가 시작돼 나머지 커밋은 이 빌드에 반영되지 않음
        - ▪ 옵션을 선택했다면 첫 번째 커밋이 이뤄지고 지정된 시간 동안 빌드 큐에서 기다리다가 거의 동시에 발생한 모든 커밋을 포함해 빌드가 시작되는 것
        - ▪ 옵션을 선택하면 빌드 대기 시간(초 단위)을 설정하는 필드인 Number of seconds가 나타남
        - ▪ 여기에 5를 입력하면 빌드가 시작되기 전에 5초 동안 빌드 큐에서 대기

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ Retry Count
        - ▪ 옵션이 설정되지 않은 상태에서 깃 리포지터리 같은 SCM(Source Code Management) 시스템을 사용하도록 작업을 구성했다면 첫 번째 체크 아웃 시도가 실패하자마자 바로 작업 실패로 처리
        - ▪ 옵션을 사용하면 작업 실패로 표시하기 전에 몇 번 더 체크아웃을 시도할 지를 정할 수 있음
        - ▪ SCM checkout retry count 값을 3으로 지정하면 Jenkins는 체크 아웃을 3 번 시도하며 재시도 사이의 시간 간격은 10초

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ 업스트림 프로젝트가 빌드 중일 때 빌드 차단
        - ▪ Jenkins에서는 작업 A에서 생성된 아티팩트가 작업 B의 작업에 사용되도록 2개의 작업을 구성할 수 있음
        - ▪ 작업 B를 작업 A에 종속되게 구성하는 것
        - ▪ 작업 A는 자바 애플리케이션을 컴파일해 .CLASS 파일을 생성하고 작업 흐름 B는 클래스 파일을 사용해 .jar 파일을 생성하는 경우 작업 A가 작업 B의 upstream job이 되고 작업 B는 작업 A의 downstream job
        - ▪ Block build when upstream project is building을 선택하면 업스트림 프로젝트(현재 작업이 의존하는 프로젝트)가 빌드 큐에 있을 때 Jenkins가 이 프로젝트 빌드 작업을 실행하지 않음
        - ▪ 작업 A가 빌드 큐에 있으면 Jenkins는 작업 B를 시작하지 않음
      - ◆ 다운스트림 프로젝트가 빌드 중일 때 빌드 차단
        - ▪ 작업 B가 빌드 큐에 있으면 Jenkins는 작업 A를 시작하지 않음

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ Use Custom workspace(사용자 빌드 경로 사용)
        - ▪ Jenkins 작업이 소스 코드를 리포지터리에서 체크아웃하는 경우에 빌드가 시작되면 기본적으로 ${JENKINS_HOME}₩workspace 폴더에 실행 중인 작업의 이름으로 워크스페이스 디렉토리가 생성됨
        - ▪ 현재 작업의 아티팩트를 다른 작업이 사용하고 있고 이 경로가 다른 작업에서 하드코딩 돼 바꿀 수가 없다면 현재 작업의 워크스페이스 위치를 변경할 수 있음
      - ◆ 표시 이름 지정
        - ▪ 웹 화면에 보여지는 프로젝트 이름 설정
      - ◆ Keep the build logs of dependencies
        - ▪ 현재 작업과 관련된 모든 빌드가 로그 순환 기능에서 제외됨
        - ▪ 로그 순환은 Jenkins의 빌드 로그를 자동으로 압축하고 삭제하는 기능

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ 소스 코드 관리
        - ▪ Git 옵션은 깃 플러그인이 설치된 경우에만 나타남
        - ▪ 일반적으로 신규 소프트웨어 빌드를 생성하는 Jenkins 작업은 중앙 리포지터리로 커밋된 최신 코드에서 작동하기 때문에 Jenkins가 최신 코드를 다운로드해 빌드할 수 있도록 깃 코드 리포지터리 URL에 이 필드를 추가해야 함
        - ▪ 공개 깃 리포지터리는 인증이 필요 없음
        - ▪ 개인 리포지터리의 경우에는 일반적으로 깃 리포지터리 구성에 필요한 사용자 이름 과 비밀번호 또는 SSH 개인 키나 API 토큰 같은 인증 정보를 제공해야 하는데 이는 Credential을 이용해서 설정
        - ▪ Branches to build 필드에 브랜치 이름을 지정

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ 빌드 트리거: 작업을 시작하는 시점을 설정
        - ▪ 빌드를 원격으로 유발
          - 스크립트를 이용해서 원격으로 빌드
          - Jenkins 작업을 원격으로 시작할 수 있는 URL: <Jenkins_URL>/job/<작업_이름>/build?token=<토큰_이름>
        - ▪ Build after other projects are built: 다른 작업을 완료한 후에 작업을 시작하고자 할 때 선택
          - 빌드가 안정한 경우에만 시작
          - 빌드가 불안정한 경우에도 시작
          - 빌드가 실패해도 시작
        - ▪ Build periodically: 윈도우의 작업 스케줄러나 유닉스 시스템의 cron 작업과 같은 기능을 제공

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ 빌드 트리거: 작업을 시작하는 시점을 설정
        - ▪ GitHub hook trigger for GITScm polling: Jenkins에서 GitHub Webhook을 통해 코드 변경 시 자동 빌드 트리거를 활성화하기 위해 사용하는 옵션
        - ▪ Poll SCM: 일정 주기마다 GitHub 폴링

항목	Poll SCM	GitHub hook trigger
작동 방식	일정 주기마다 GitHub 폴링	Webhook이 Jenkins에 알림
효율성	불필요한 요청 발생 가능	이벤트 기반으로 즉시 트리거
추천 여부	테스트나 내부 네트워크에서만 적합	GitHub hook trigger 권장

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ Environment
        - ▪ Delete workspace before build starts: 빌드 실행 전에 워크스페이스(workspace) 디렉토리를 완전히 삭제하여 이전 빌드에서 남은 파일이나 캐시가 새 빌드에 영향을 주지 않도록 하는 기능
        - ▪ Use secret text(s) or file(s): 민감한 값(비밀번호, API 토큰, 인증서 파일 등)을 빌드 환경에 안전하게 주입하기 위한 기능으로 Credentials Binding Plugin을 설치해야 사용할 수 있음
        - ▪ Add timestamps to the Console Output: 각 빌드 로그 줄 앞에 시간 정보가 표시되어 디버깅이나 성능 분석 시 매우 유용
        - ▪ Inspect build log for published build scans: Gradle 또는 Maven 빌드에서 Build Scan 기능을 사용했을 때, 빌드 로그에서 Build Scan URL을 확인하는 과정을 의미
        - ▪ Terminate a build if it's stuck: 빌드가 특정 시간 이상 실행되면 자동으로 강제 종료(Abort) 시켜 리소스 낭비나 빌드 대기열 문제를 방지하는 기능
        - ▪ with Ant: Freestyle 프로젝트나 Pipeline에서 Ant를 실행하여 Java 프로젝트를 빌드할 때 활용하는데 이를 위해 먼저 Jenkins에 Ant를 설정하고, 이후 Job에서 Invoke Ant 또는 withAnt 스텝을 사용

Free Style Job

- ❖ Free Style Job 생성 및 빌드
  - ✓ 작업 생성
    - ● 작업 구성
      - ◆ Build Steps
        - ▪ Build Steps 섹션은 1개 이상의 스탭을 하나씩 실행하는 식으로 현재 작업에 할당된 태스크를 수행할 때 사용
        - ▪ 스텝은 배치 파일을 실행할 수도 있고 다른 빌드 도구를 실행하는 것일 수도 있음
        - ▪ 윈도우 터미널의 명령을 실행하는 스텝을 추가한다면 Add build step 버튼을 클릭한 후 Execute Windows batch command 옵션을 선택하고 Command 필드에 터미널 명령어를 추가
        - ▪ 스텝을 제거하고 싶다면 오른쪽에 위치한 빨간색 x 버튼을 클릭
        - ▪ 1개의 작업에는 여러 개의 스텝을 추가할 수 있으며 여러 개의 스텝은 그림 위에서부터 순차적으로 실행
      - ◆ 포스트-빌드 액션
        - ▪ 포스트-빌드 액션 섹션은 할당된 태스크가 완료된 이후에 수행하려는 작업이 있을 때 사용
        - ▪ 작업 완료 후 이메일 알림을 보내는 작업이 대표적
        - ▪ 액션을 추가하려면 Acid Post-Build Action 옵션을 클릭한 후 필요한 옵션을 선택

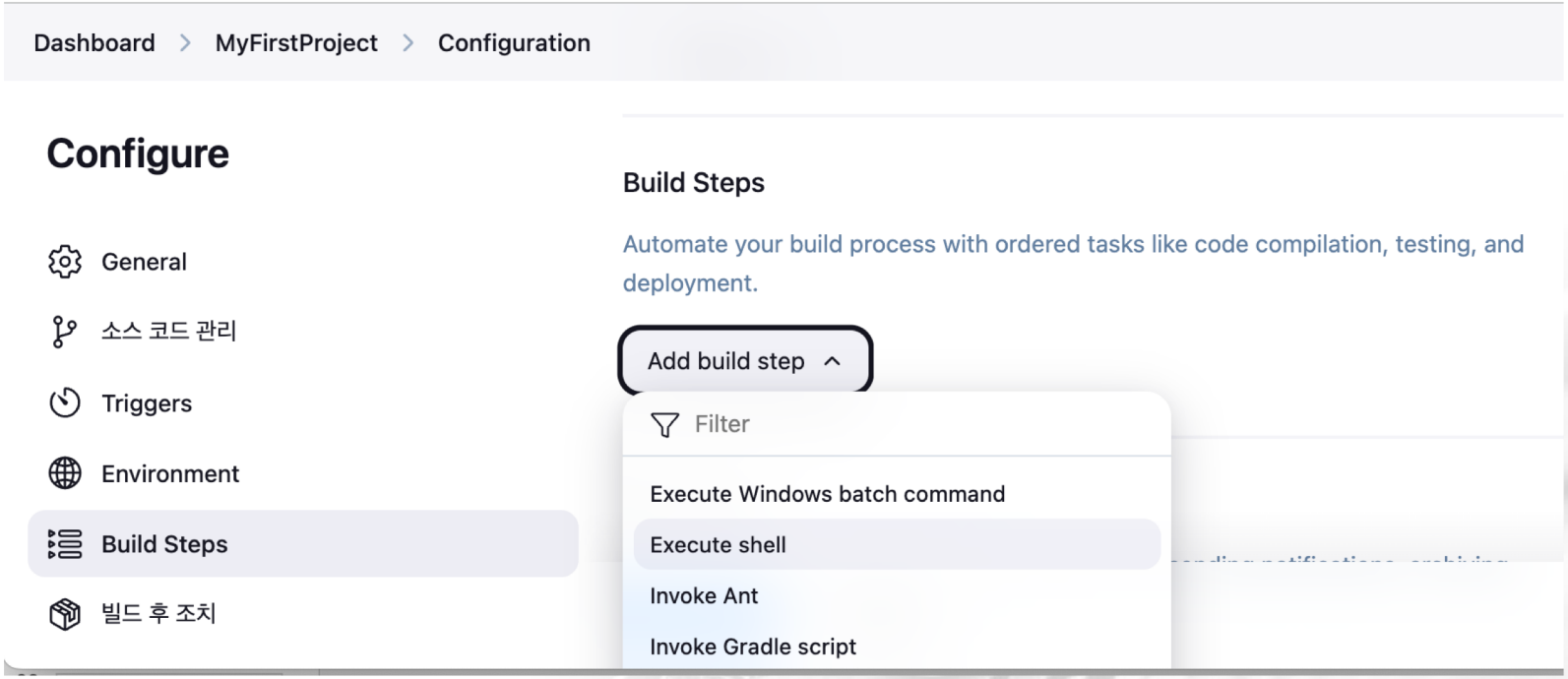
Free Style Job

❖ 샘플 프로젝트 생성 및 빌드

✓ 작업 생성

- 작업 이름을 입력
- Freestyle project 선택 후 OK

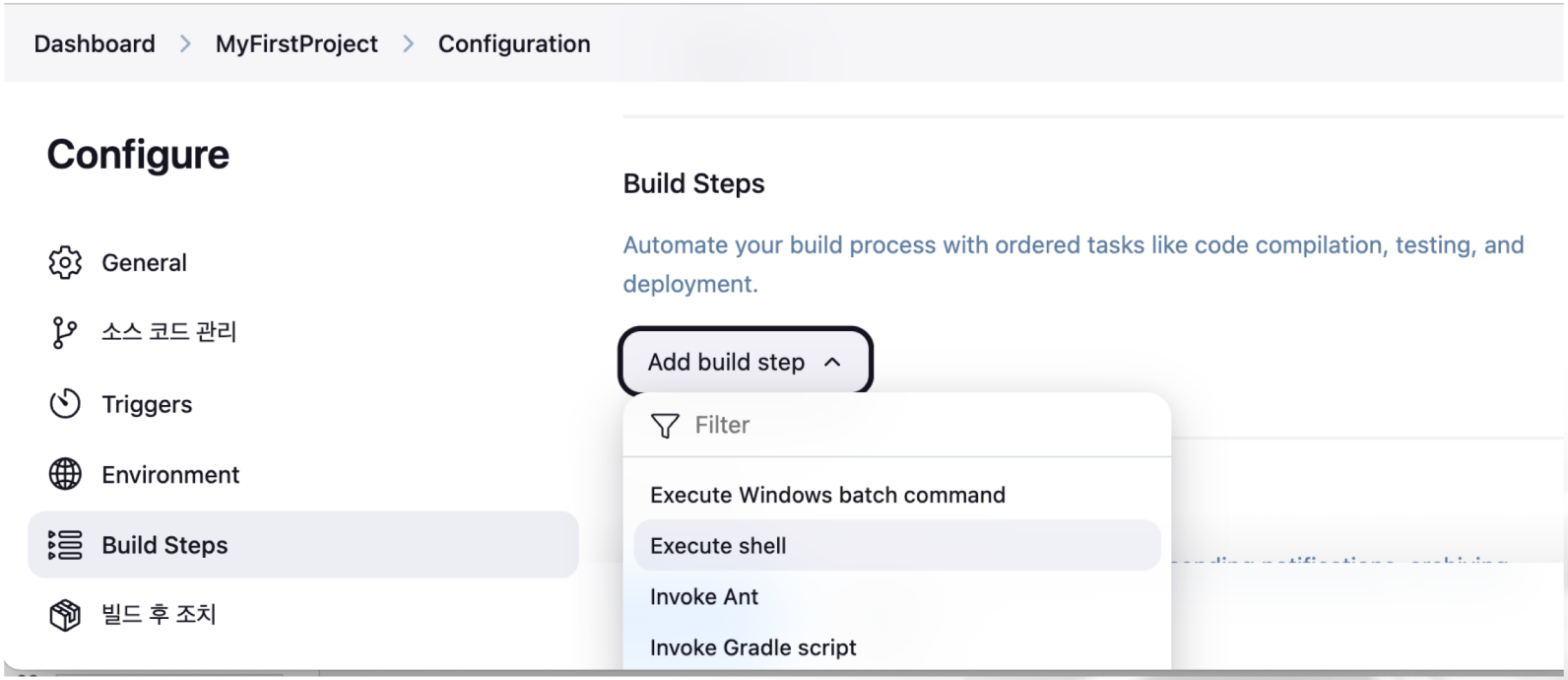
✓ Build Steps 항목에서 Add build step -> Execute Shell 선택



Free Style Job

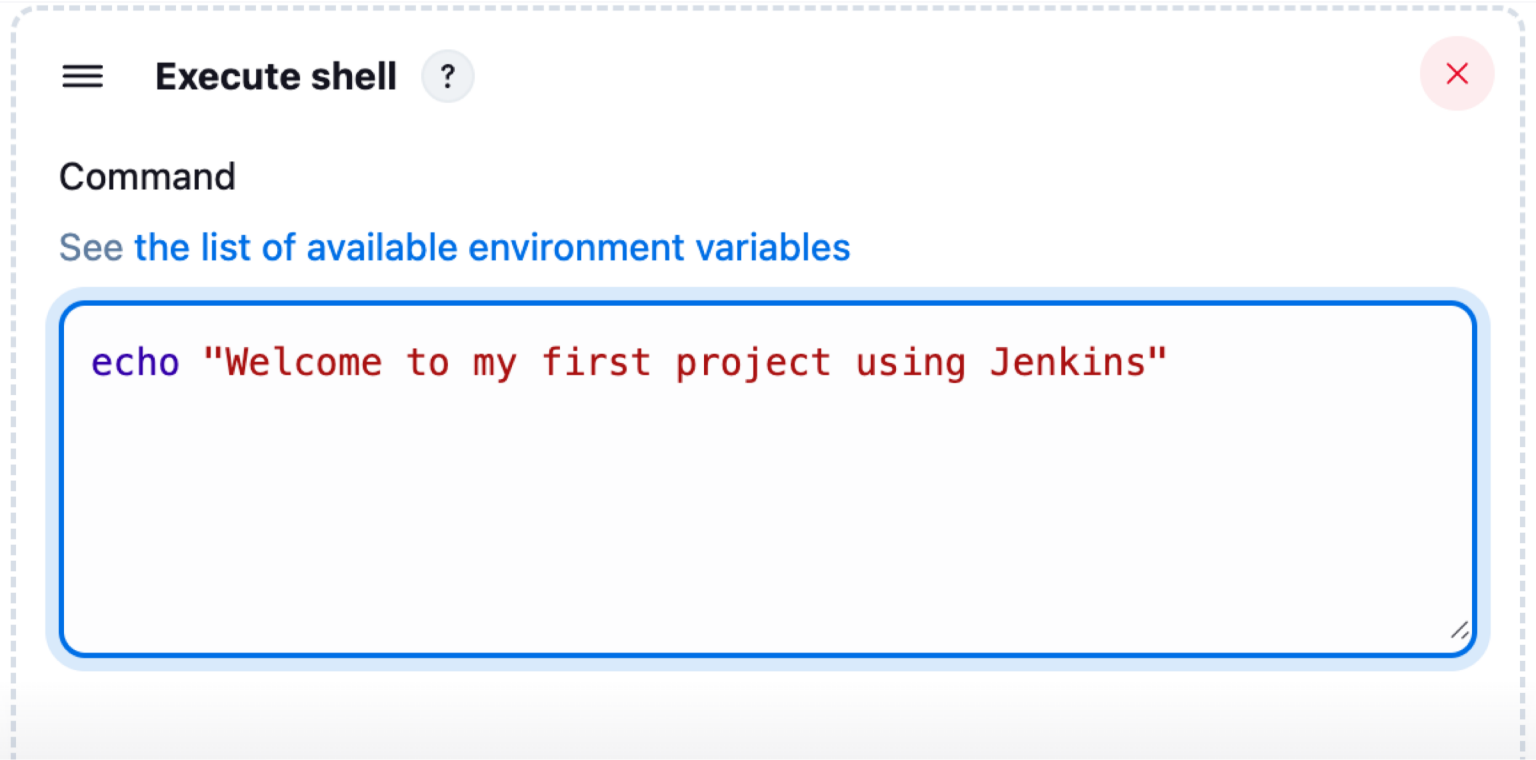
❖ 샘플 프로젝트 생성

✓ Build Steps 항목에서 Add build step -> Execute Shell 선택



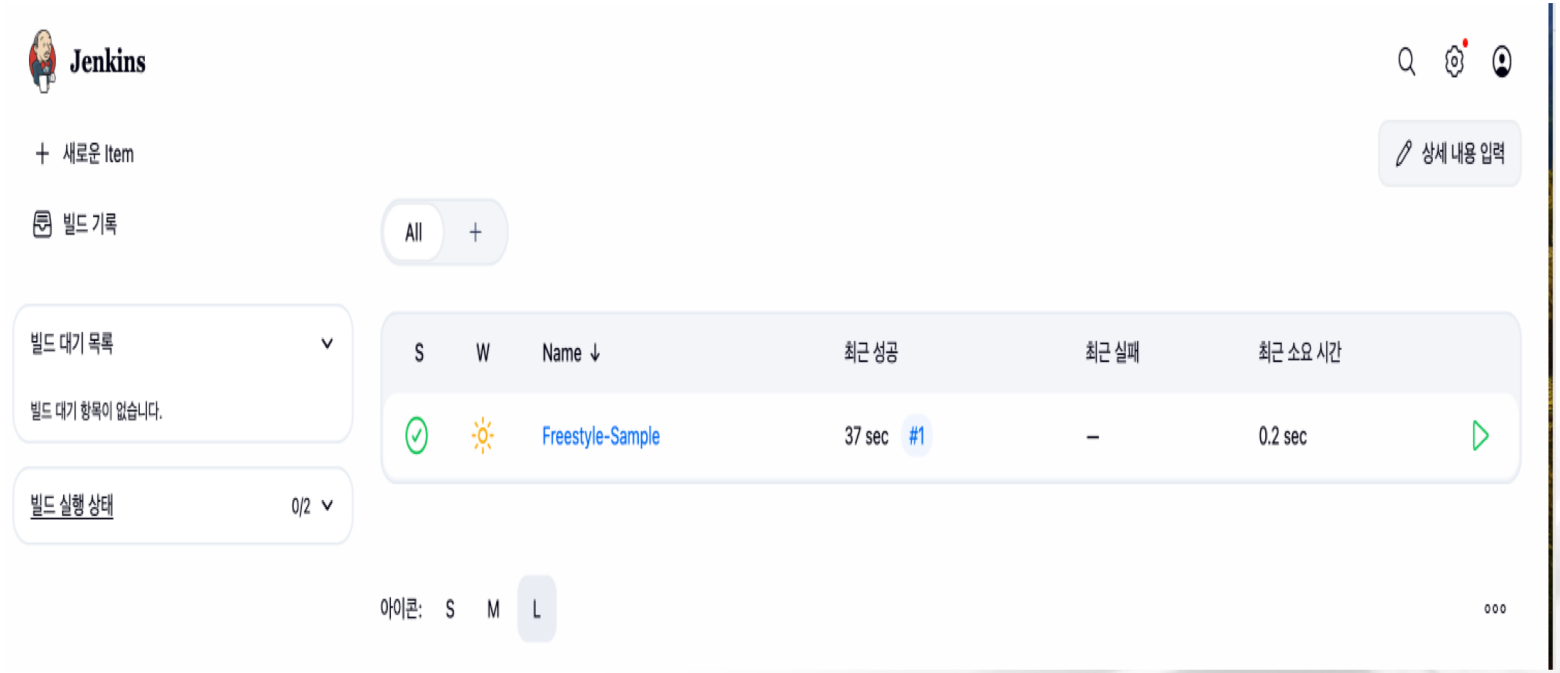
Free Style Job

- 샘플 프로젝트 생성
  - ✓ Execute Shell에서 명령 입력



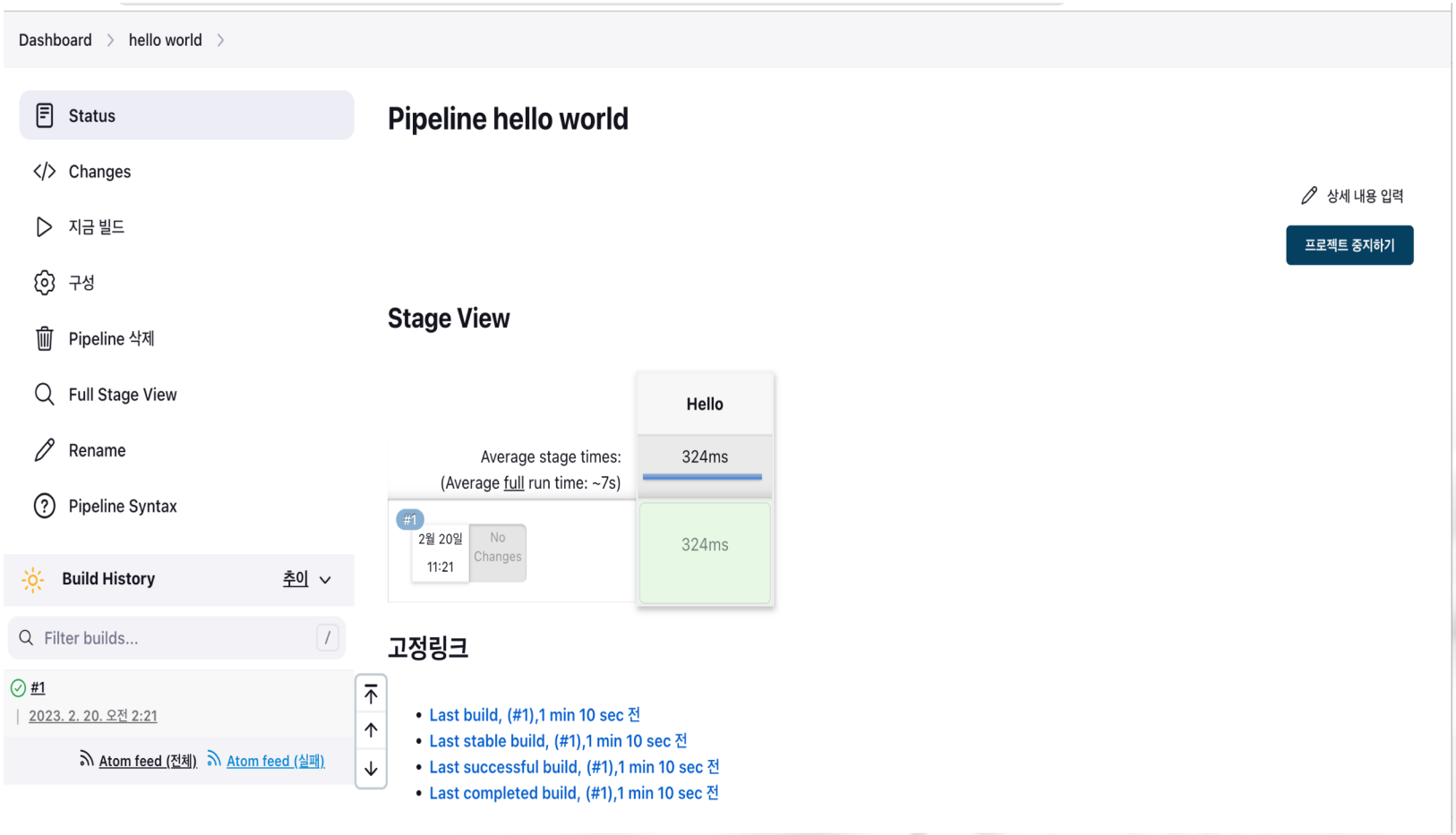
Free Style Job

- 샘플 프로젝트 생성
  - ✓ 실행: 실행 아이콘 클릭



Free Style Job

- 샘플 프로젝트 생성
  - ✓ 실행: 상세보기 화면에서 Build Now 클릭



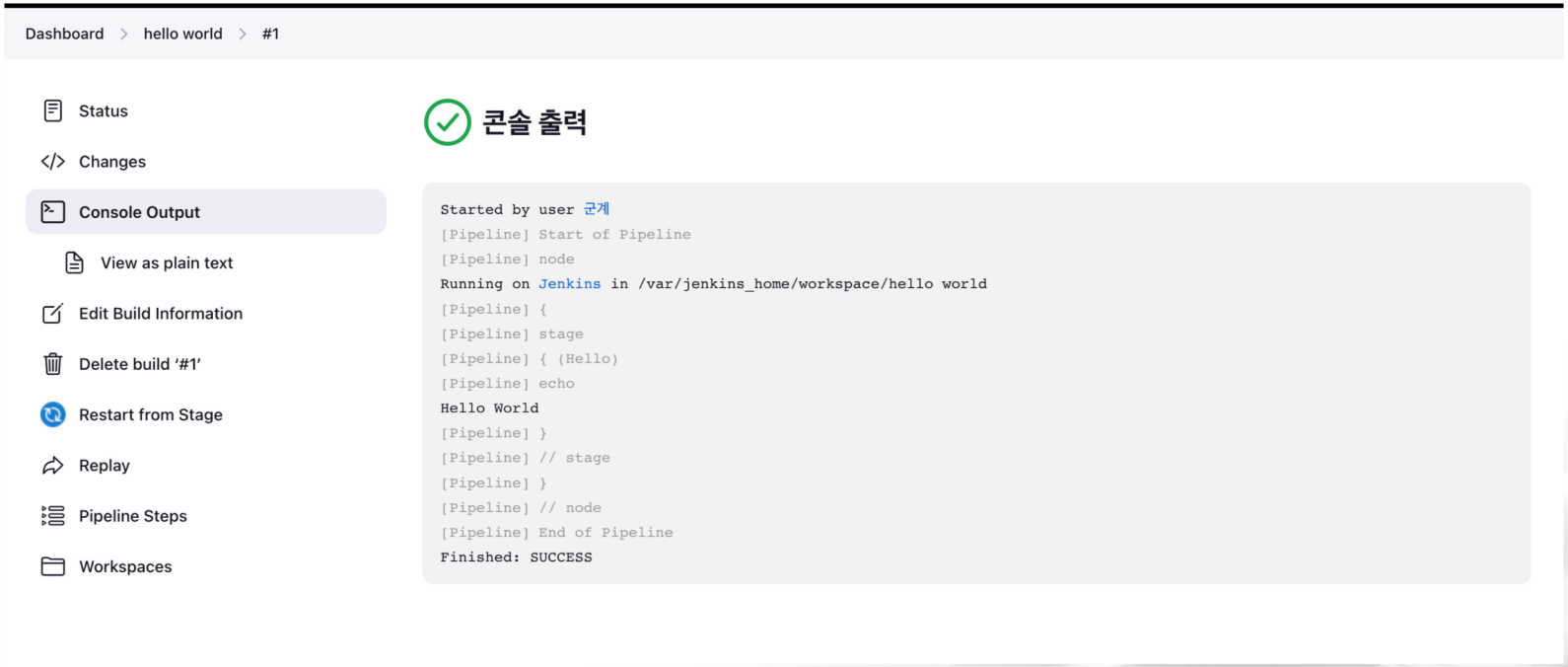
Free Style Job

- ❖ 샘플 프로젝트 생성
  - ✓ 작업 결과 확인
    - 작업이 완료되면 대시 보드 페이지 왼쪽에 표시된 Build History빌드 내역 링크를 클릭해서 빌드 실행 내역을 볼 수 있음

Free Style Job

❖ 샘플 프로젝트 생성

✓ 세부 빌드 내역 확인: 왼쪽 화면에서 현재 빌드 된 항목을 클릭하고 Console Output을 선택해서 명령어가 제대로 수행되는지 확인



Free Style Job

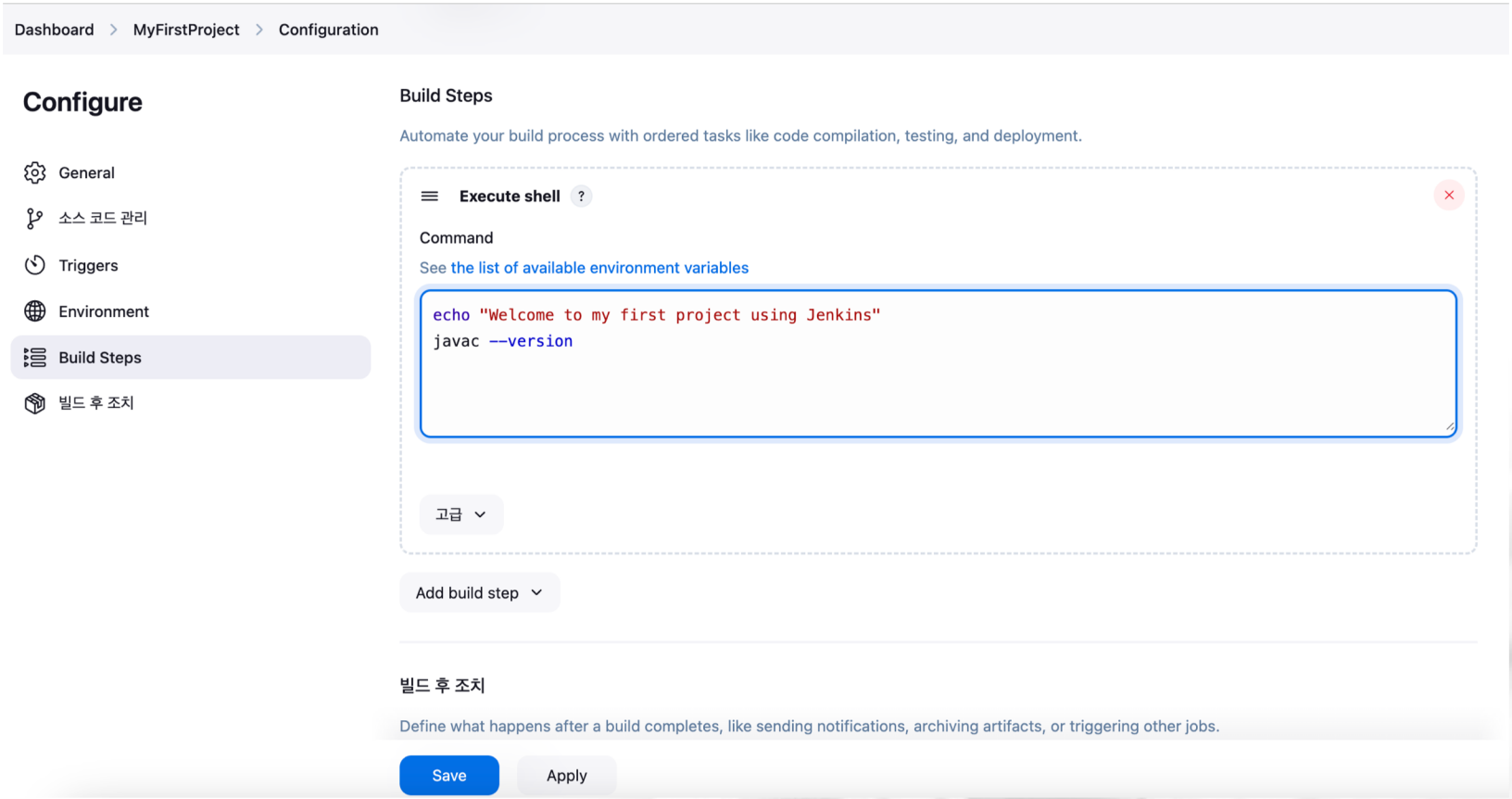
❖ 샘플 프로젝트 수정

✓ 대시보드의 프로젝트 화면으로 돌아가서 프로젝트를 선택하고 구성(Configure)을 클릭

Free Style Job

❖ 샘플 프로젝트 수정

✓ Build Step을 눌러서 코드를 추가



Free Style Job

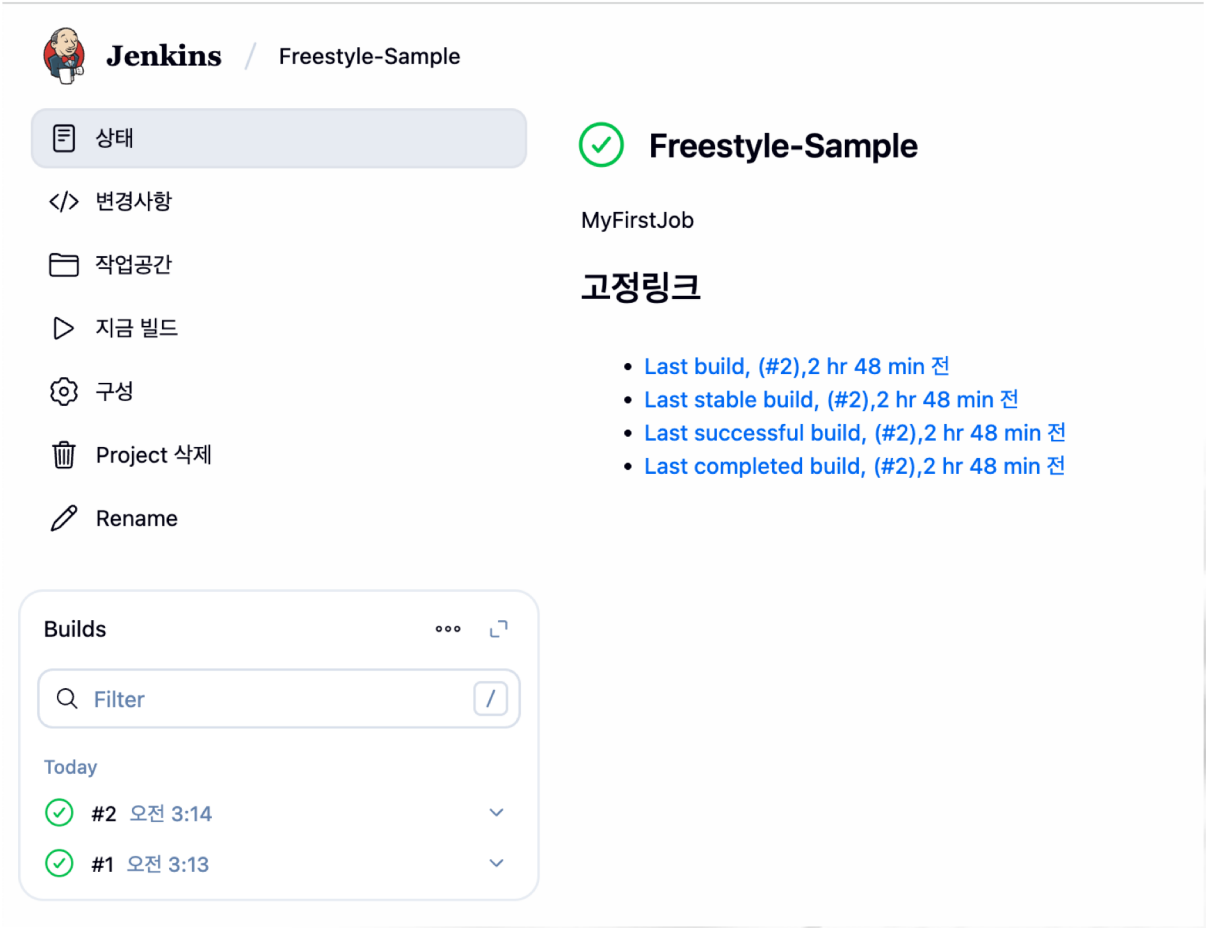
❖ 샘플 프로젝트 수정

✓ 다시 빌드를 수행한 후 빌드 된 결과물을 선택하고 Console Output을 확인

Free Style Job

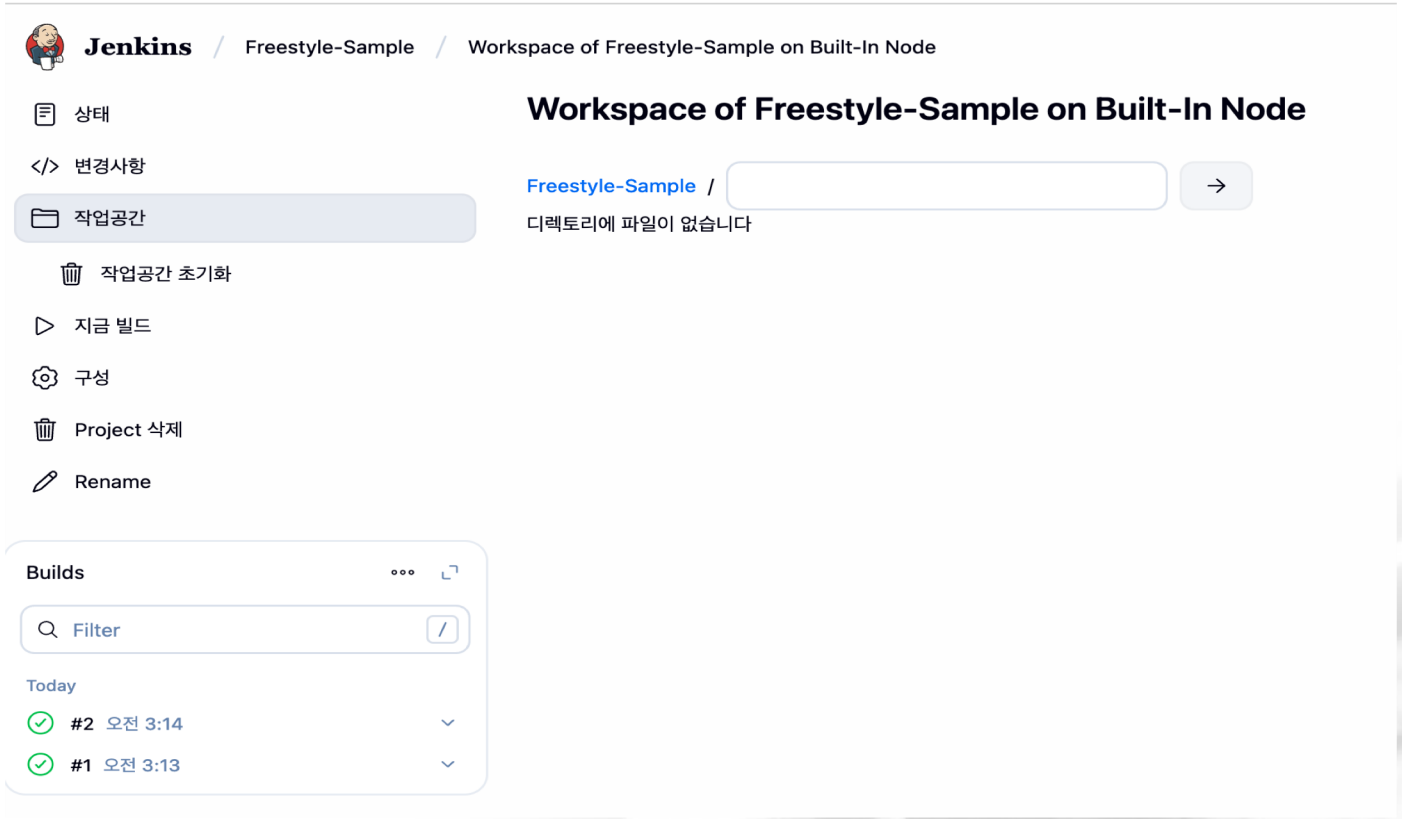
❖ 결과물 확인

✓ 프로젝트 상태 확인



Free Style Job

- ❖ 결과물 확인
  - ✓ 프로젝트 내부 구조 확인(/var/lib/jenkins/workspace)



Free Style Job

- ❖ 결과물 확인
  - ✓ Docker에 설치한 경우
    - docker exec -it jenkins-server bash
    - jenkins@89b2614399fd:/$ cd /var/jenkins_home/workspace
    - jenkins@89b2614399fd:~/workspace$ ls -al

      total 12

      drwxr-xr-x 3 jenkins jenkins 4096 Dec 18 23:17 .

      drwxr-xr-x 14 jenkins jenkins 4096 Dec 18 23:36 ..

      drwxr-xr-x 2 jenkins jenkins 4096 Dec 18 23:17 MyFirstProject

      jenkins@89b2614399fd:~/workspace$
    - jenkins@89b2614399fd:~/workspace$ cd MyFirstProject
    - jenkins@89b2614399fd:~/workspace/MyFirstProject$ ls -al

      total 8

      drwxr-xr-x 2 jenkins jenkins 4096 Dec 18 23:17 .

      drwxr-xr-x 3 jenkins jenkins 4096 Dec 18 23:17 ..

자격 증명

- 개요
  - ✓ CI/CD 자동화 서버인 젠킨스에서 작업을 수행하려면 다양한 외부 도구들(넥서스 아티팩트 리포지터리, 깃 소스 코드 리포지터리 등)에 접속해야 하는데 이러한 외부 리포지터리들의 인증 방식에는 사용자 이름과 비밀번호를 사용하는 기본 인증, 개인 키와 공개 키를 사용하는 SSH 인증, 접속 요청 시 API 토큰을 전송하는 API 토큰 기반 인증 등 다양한 방식이 있음
  - ✓ 젠킨스에서 이들 도구에 접속할 때는 적절한 인증 정보를 제공해야 함
  - ✓ 젠킨스가 접속하려는 시스템에서 기본 인증을 요구한다면 사용자 이름과 비밀번호를 제공해야 하고 SSH 인증을 요구한다면 개인 키를 제공해야 함
  - ✓ 젠킨스에서는 인증에 필요한 정보를 credential(자격 증명)이라고 함
  - ✓ 젠킨스에서는 자격 증명 항목을 생성하고 인스턴스 내에 이 정보를 저장할 수 있음

자격 증명

- 개요
  - ✓ CI/CD 자동화 서버인 젠킨스에서 작업을 수행하려면 다양한 외부 도구들(넥서스 아티팩트 리포지터리, 깃 소스 코드 리포지터리 등)에 접속해야 하는데 이러한 외부 리포지터리들의 인증 방식에는 사용자 이름과 비밀번호를 사용하는 기본 인증, 개인 키와 공개 키를 사용하는 SSH 인증, 접속 요청 시 API 토큰을 전송하는 API 토큰 기반 인증 등 다양한 방식이 있음
  - ✓ 젠킨스에서 이들 도구에 접속할 때는 적절한 인증 정보를 제공해야 함
  - ✓ 젠킨스가 접속하려는 시스템에서 기본 인증을 요구한다면 사용자 이름과 비밀번호를 제공해야 하고 SSH 인증을 요구한다면 개인 키를 제공해야 함
  - ✓ 젠킨스에서는 인증에 필요한 정보를 credential(자격 증명)이라고 함
  - ✓ 젠킨스에서는 자격 증명 항목을 생성하고 인스턴스 내에 이 정보를 저장할 수 있음

자격 증명

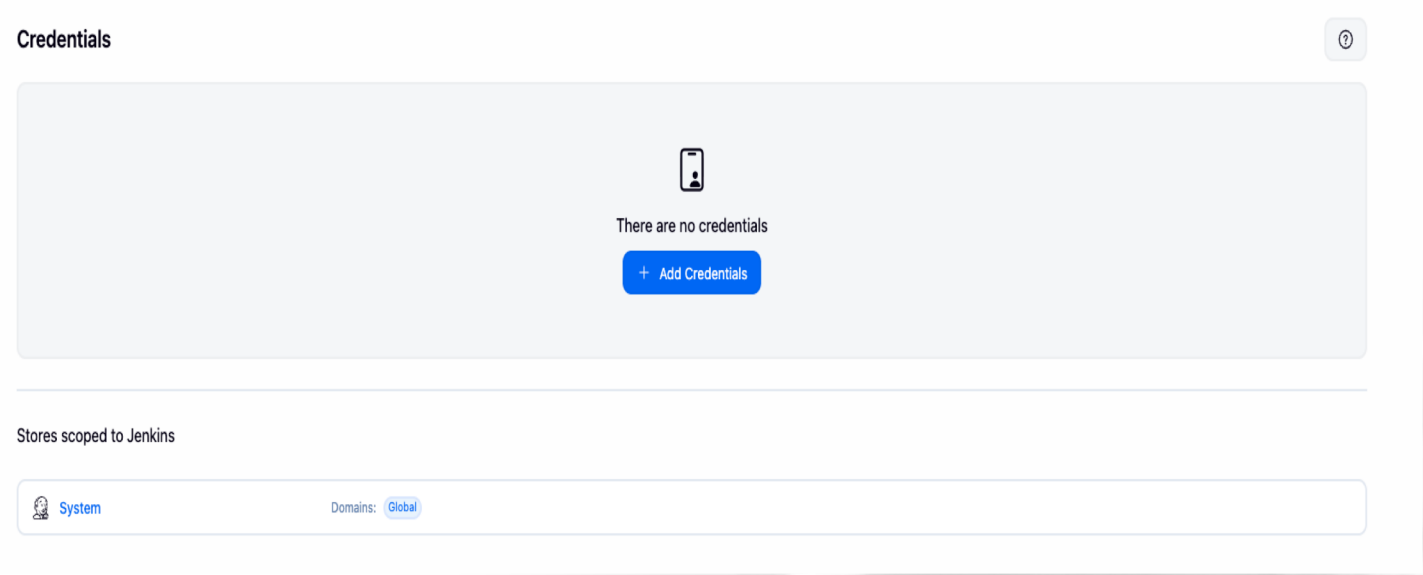
- 자격 증명 생성
  - ✓ 자주 사용하는 기법
    - 기본 인증
    - SSH 인증
    - API Token
    - Certificate(인증서)
  - ✓ 범위(Scope)
    - Global(전역): 모든 젠킨스 작업과 서버 시스템에서 사용 가능
    - System: 젠킨스 인스턴스에서 이메일 인증이나 에이전트 연결 등과 같은 시스템 기능을 수행하는 데만 사용할 수 있는 것으로 이 자격 증명은 젠킨스 작업에서는 사용할 수 없음

자격 증명

- 자격 증명 생성
  - ✓ Domain
    - 유사한 시스템들에서 사용하는 자격 증명을 그룹화하는 방법
    - 자격 증명 항목에 별도의 설정이 없었다면 기본 도메인에 생성되는데 예를 들어 10개의 자격 증명 항목이 있다고 가정해보고 이 중 3개는 깃랩 코드 리포지터리에 접속하는 데 사용되고 그 외에 7개는 AWS와 같은 다른 시스템에 접속하기 위한 것 인 경우 만약 사용자가 젠킨스 작업에서 깃랩 코드 리포지터리에 접속하는 구성을 한다면 10개의 모든 자격 증명 항목이 표시되기 때문에 사용 가능한 자격 증명 항목 중에서 원하는 항목을 선택하는 것이 어려움
    - GitLab Credentials 라는 도메인을 만들고 생성된 도메인 아래에 3개의 자격 증명 항목을 배치하면 깃랩에서 코드 리포지터리에 대한 액세스를 구성할 때 10 개의 자격 증명 항목이 표시되지 않고 3개의 항목만 표시됨

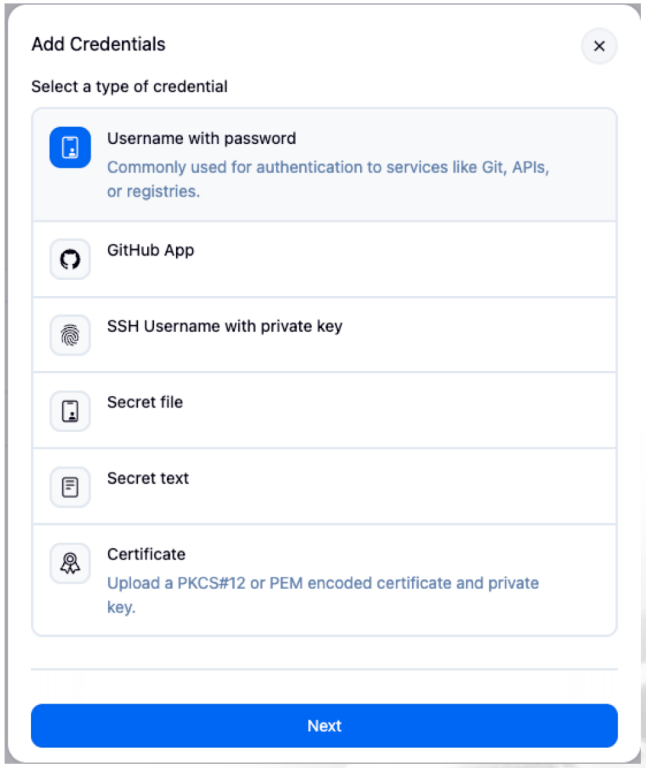
자격 증명

- ❖ 자격 증명 생성
  - ✓ 전역 범위 와 전역 도메인에서 자격 증명 생성
    - 젠킨스 대시보드에서 설정 아이콘을 클릭해 Manage Jenkins 페이지로 이동
    - Configure Credentials 메뉴를 클릭



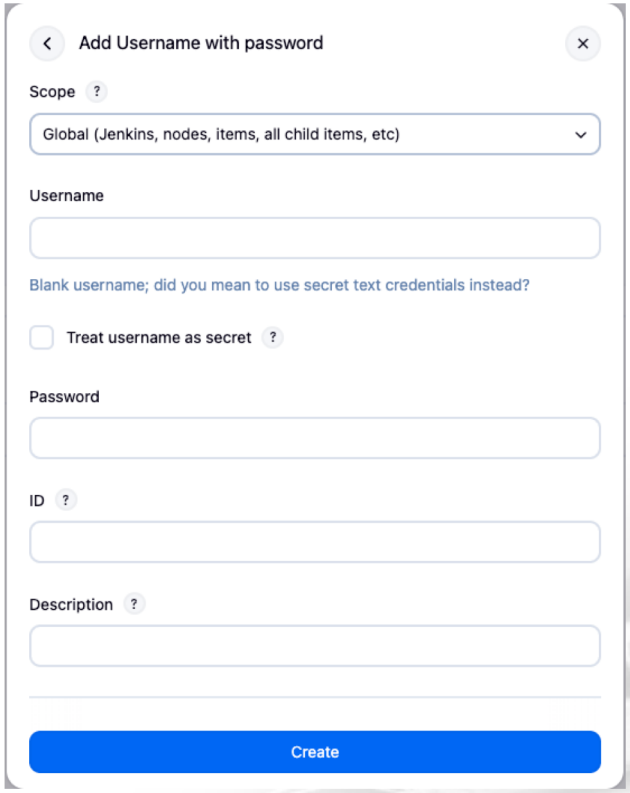
자격 증명

- ❖ 자격 증명 생성
  - ✓ 전역 범위 와 전역 도메인에서 자격 증명 생성
    - Add Credentials 클릭



자격 증명

- ❖ 자격 증명 생성
  - ✓ 전역 범위 와 전역 도메인에서 자격 증명 생성
    - Add Credentials 클릭



Maven Project

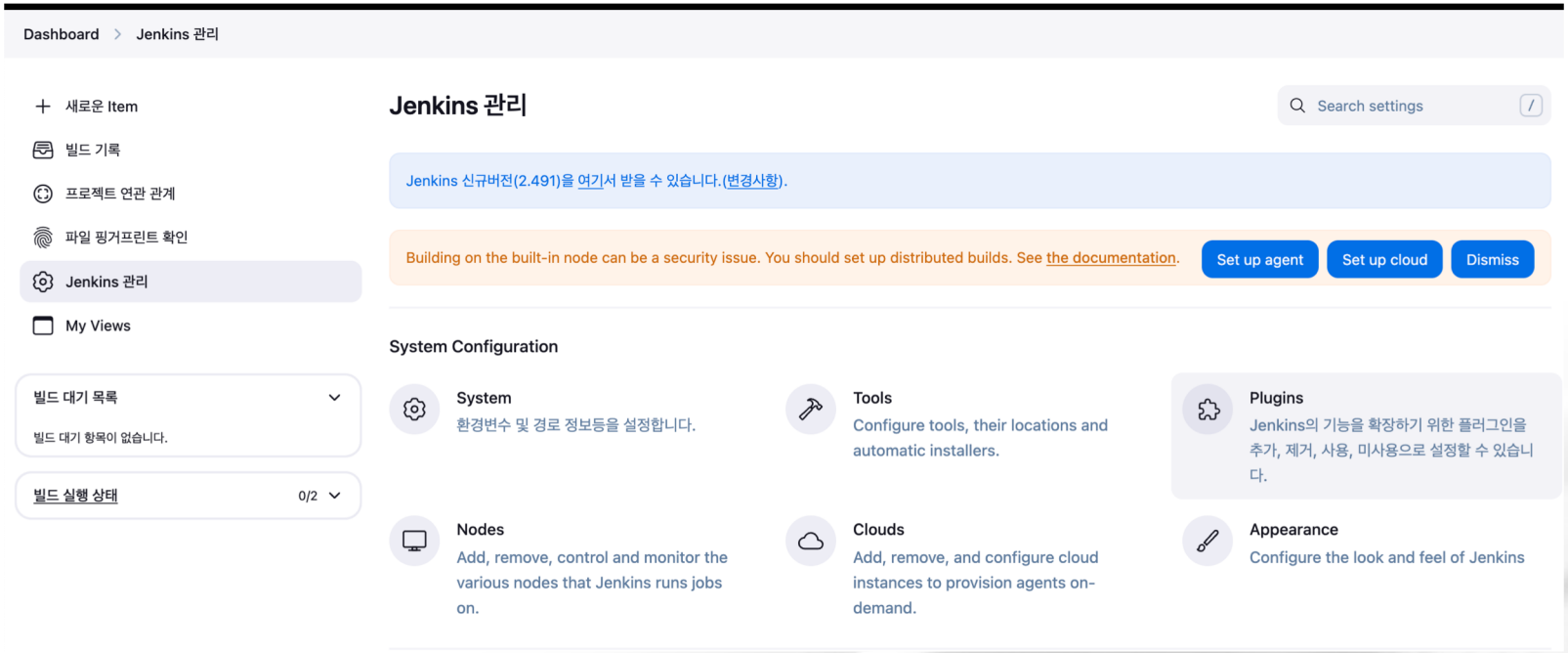
- 자바 API 프로젝트
  - ✓ 개발 과정
    - 3rd Party 라이브러리 다운로드
    - 프로젝트에 라이브러리 추가
    - 단위 테스트 케이스 작성
    - 애플리케이션과 단위 테스트 케이스 코드 컴파일
    - 테스트 케이스 실행
    - 애플리케이션 번들링/패키징
    - 아티팩트 레포지토리로 릴리스
  - ✓ 빌드 도구
    - Apache Ant
    - Maven
    - Gradle

Maven Project

❖ 설정

✓ Git 사용 설정

- Plugin 설치 확인: Jenkins 관리 -> Plugins -> Installed plugins

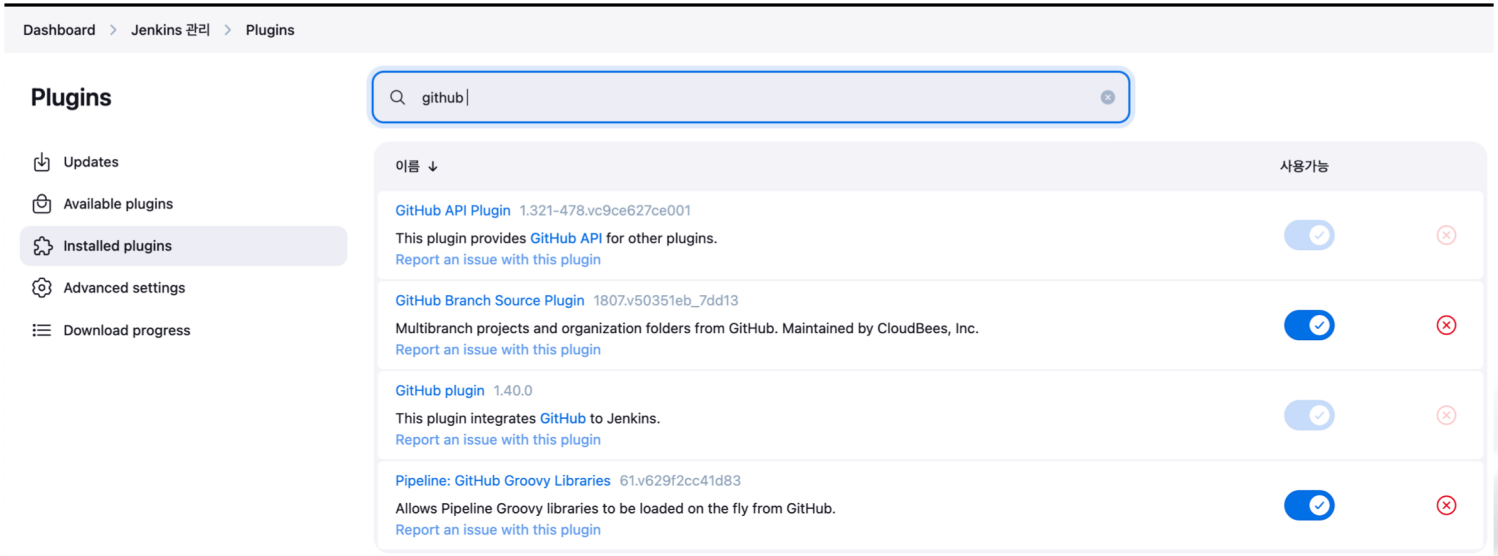


Maven Project

❖ 설정

✓ Git 사용 설정

- Plugin 설치 확인: Jenkins 관리 -> Plugins -> Installed plugins

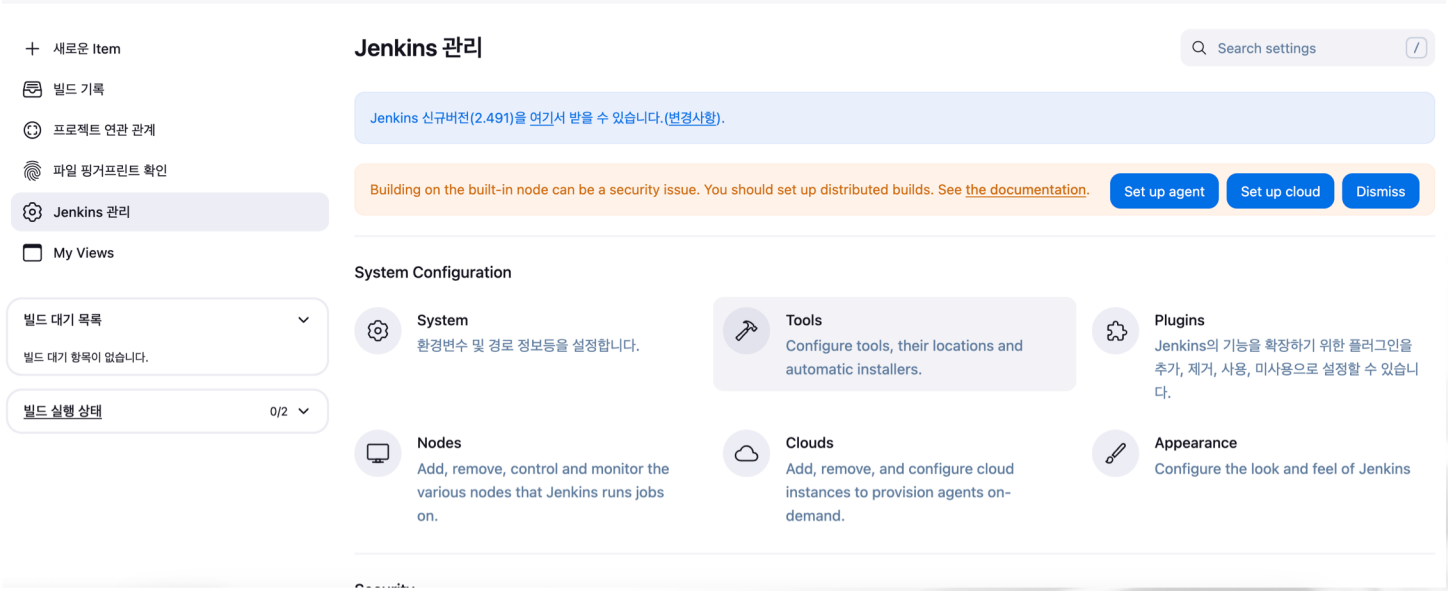


Maven Project

❖ 설정

✓ Git 사용 설정

- Git 관련 설정: Jenkins 관리 -> Tools -> Git

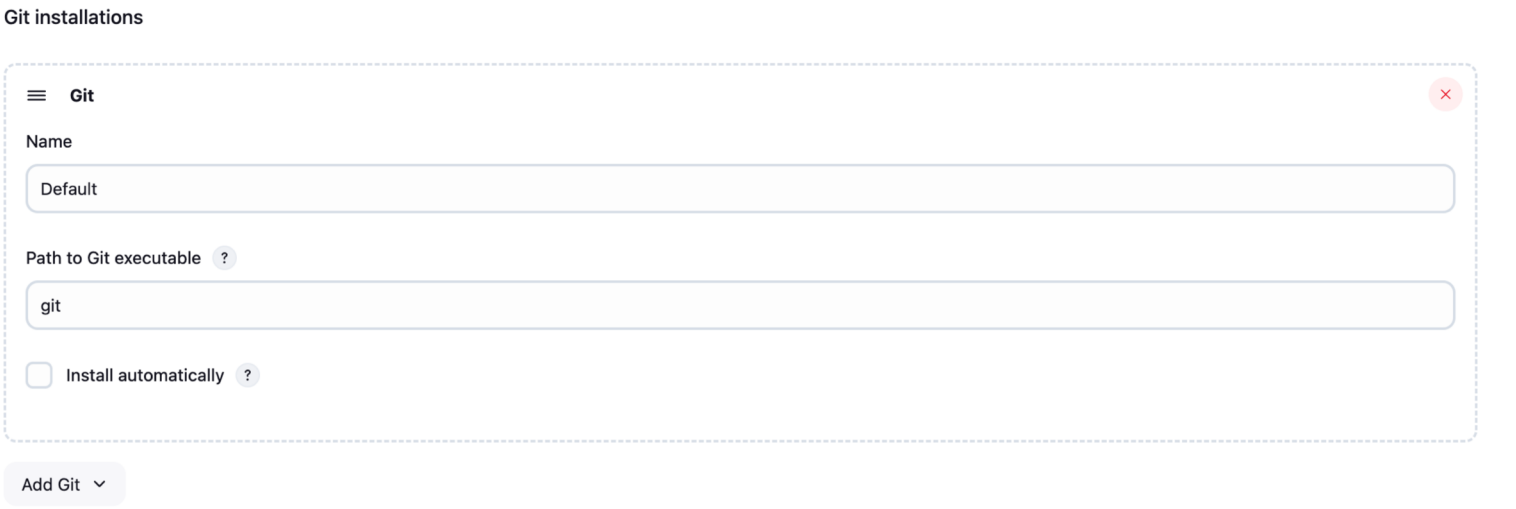


Maven Project

❖ 설정

✓ Git 사용 설정

- Git 관련 설정: Jenkins 관리 -> Tools -> Git



Maven Project

- ❖ 설정
  - ✓ Git 사용 설정
    - ● Git 명령어 사용 여부 확인

      docker ps

      CONTAINER ID IMAGE COMMAND CREATED STATUS PORTS NAMES

      89b2614399fd jenkins/jenkins "/usr/bin/tini -- /u..." 33 hours ago Up 2 hours 0.0.0.0:8080->8080/tcp, 0.0.0.0:50000->50000/tcp

      docker exec -it jenkins-server bash

      jenkins@89b2614399fd:/$ git --version

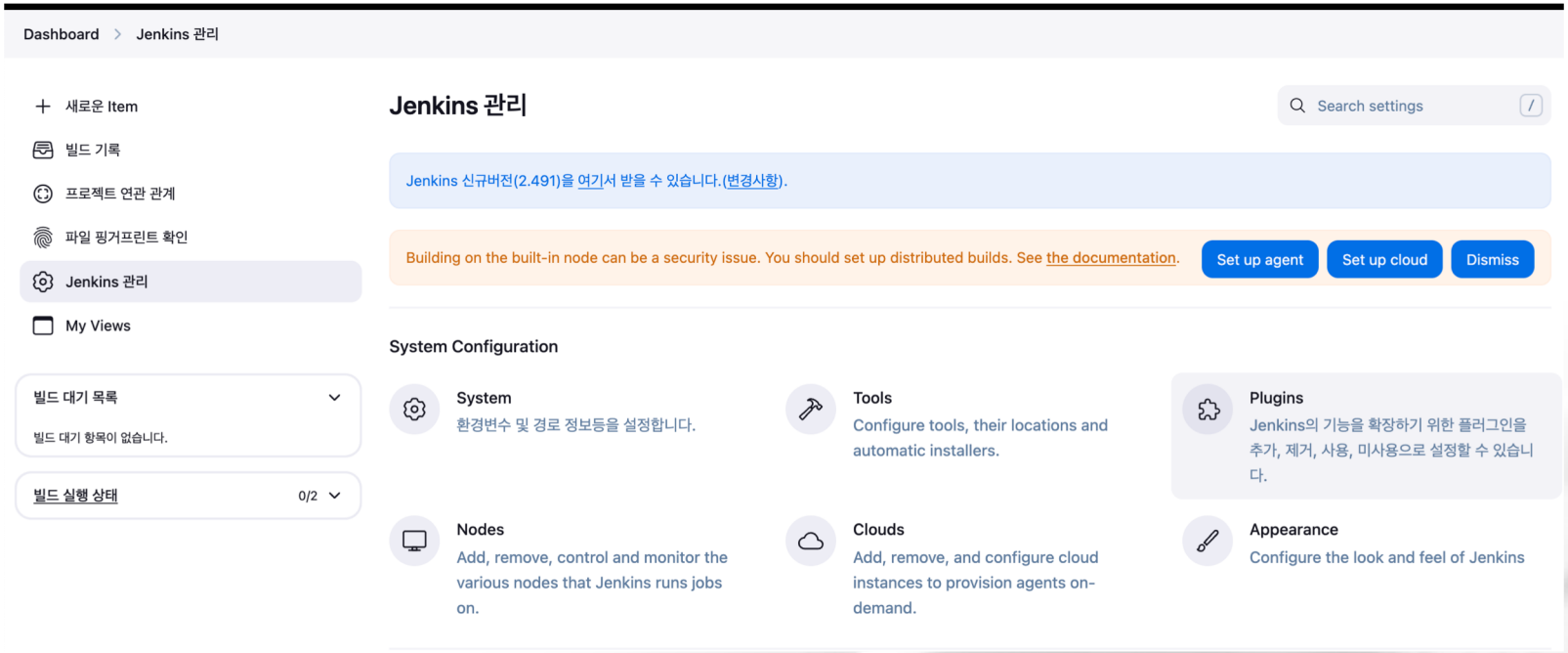
      git version 2.39.5

Maven Project

❖ 설정

✓ Maven 사용 설정

- Plugin 설치 확인: Jenkins 관리 -> Plugins -> Installed plugins

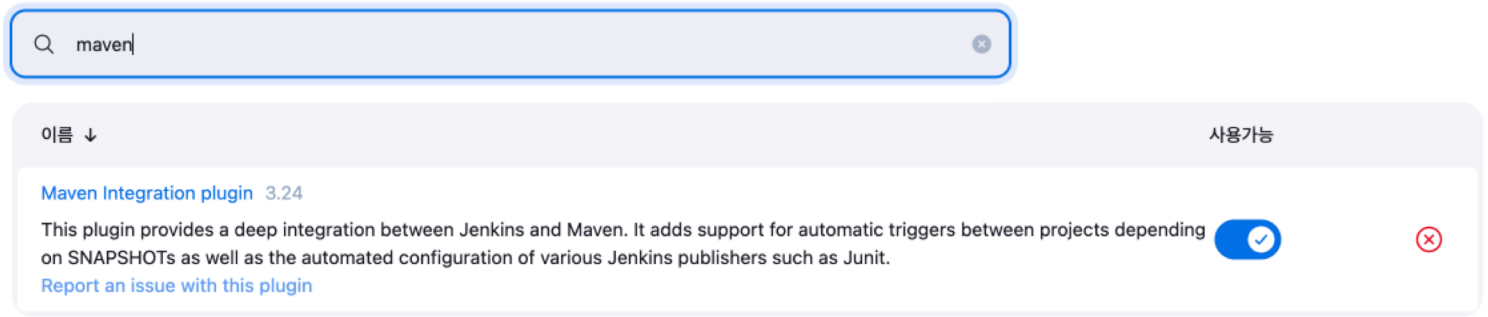


Maven Project

❖ 설정

✓ Maven 사용 설정

- Plugin 설치 확인: Jenkins 관리 -> Plugins -> Available(Installed) plugins

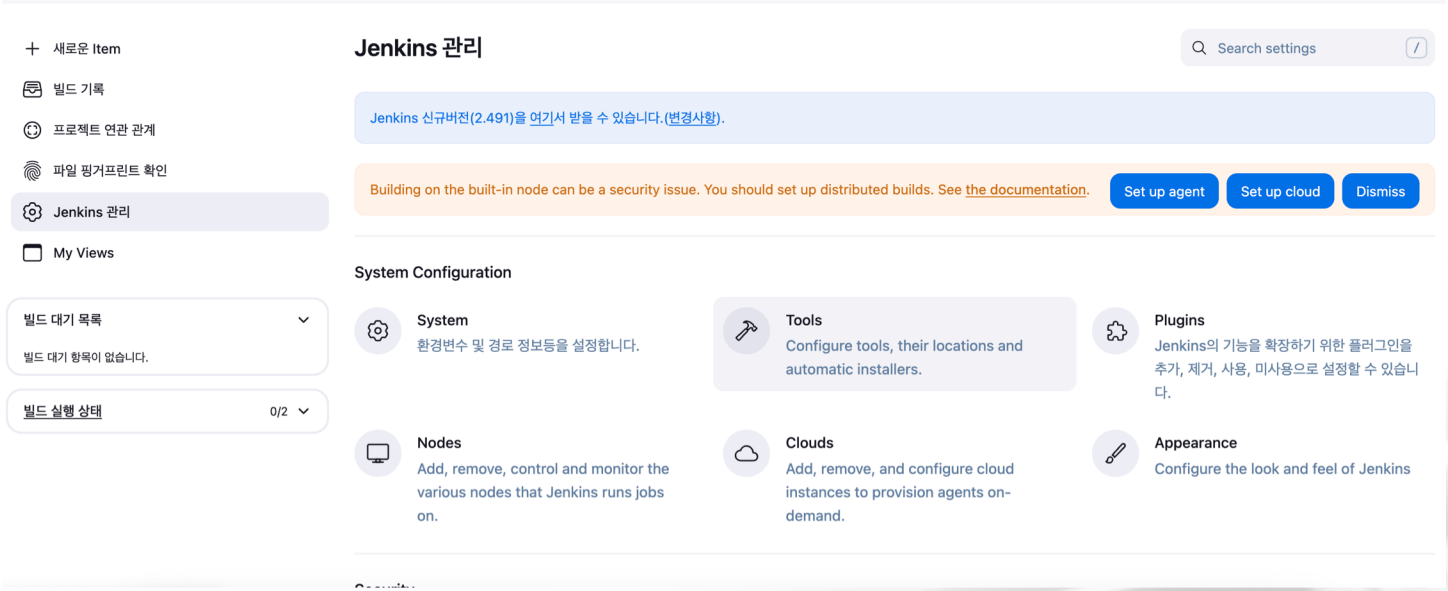


Maven Project

❖ 설정

✓ Maven 사용 설정

- Maven 관련 설정: Jenkins 관리 -> Tools -> Maven

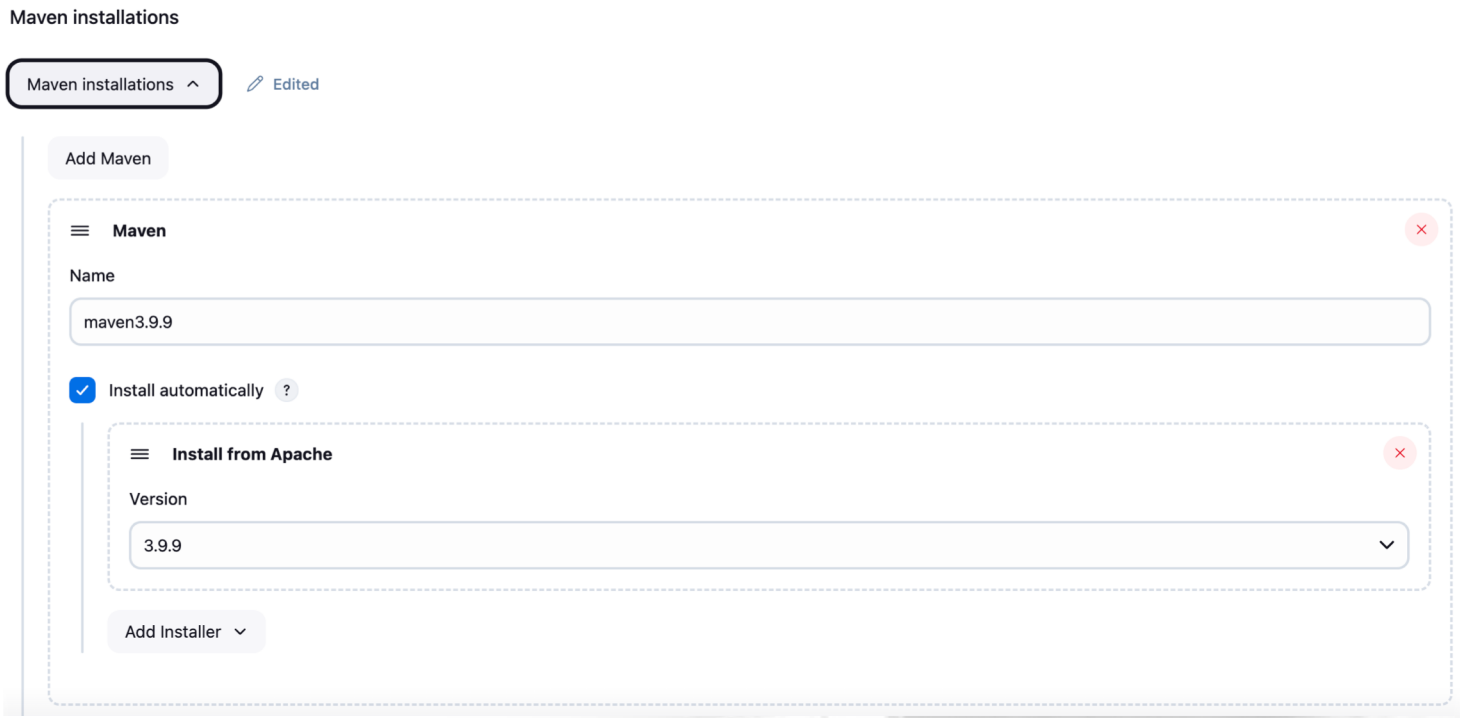


Maven Project

❖ 설정

✓ Maven 사용 설정

- Maven 관련 설정: Jenkins 관리 -> Tools -> Maven



Maven Project

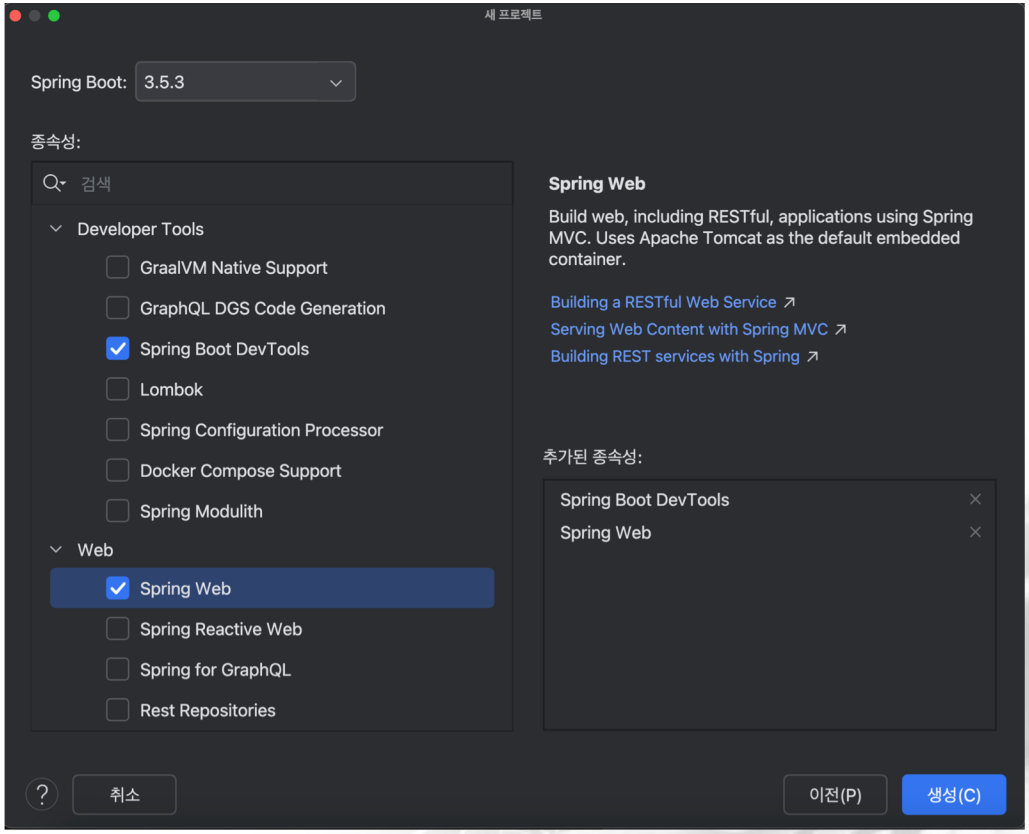
- Maven 프로젝트 빌드 수명 단계와 순서
  - ✓ Clean: 이전에 생성된 모든 컴파일 파일 과 패키지 파일을 삭제하고 새로운 빌드 수명 주기를 실행
  - ✓ 리소스 다운로드: mvnrepository.com 같은 아티팩트 레포지토리에서 프로젝트에 필요한 의존성을 다운로드하는데 메이븐 플러그인도 순환 참조를 포함한 모든 프로젝트 의존성을 다운로드
  - ✓ 애플리케이션 소스 코드 컴파일: javac를 내부적으로 사용하는 maven-copiler-plugin을 사용해 소스 코드를 컴파일하는데 컴파일러 플러그인은 기본 경로인 src/main/java 안에 있는 모든 JAVA 파일을 찾아 컴파일을 수행하고 결과물인 .class 파일을 생성하는데 이 파일은 프로젝트디렉토리/target/classes 디렉토리에 저장
  - ✓ 단위 테스트: test-compile 단계에서 src/test/java 내에 단위 테스트 코드를 컴파일하고 클래스 파일을 프로젝트디렉토리/target/test-classes 디렉토리에 저장하고 maven-surefire-plugin을 사용해 Junit 또는 TestNG 프레임워크를 실행하고 단위 테스트 케이스를 시작
  - ✓ 패키징: maven-jar-plugin은 프로젝트디렉토리/target/classes 디렉토리에 생성된 .class 파일 등의 컴파일 결과 파일을 Jar 파일로 묶는데 JAR 파일은 프로젝트디렉토리/target 디렉토리에 생성

Maven Project

- Maven 프로젝트 빌드 수명 단계와 순서
  - ✓ 릴리스
    - maven-install-plugin을 사용해 JAR 파일을 메이븐의 로컬 레포지터리에 설치
    - JAR 파일의 사용자가 원격지에 있다면 이 파일은 mvnrepository.com처럼 웹에서 접속할수 있는 중앙 리포지터리로 배포돼야 함
    - 이 과정은 deploy goal을 실행해 진행하며 mavel-deploy-plugin을 사용해 JAR 파일을 중앙 레포지터리로 배포해 개발자가 이를 다운로드하고 사용할 수 있도록 함

Maven Project

- ❖ Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - ● Spring Dev Tools 와 Spring Web 의존성을 가진 프로젝트 생성(war)



Maven Project

- ❖ Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - ● 기본 디렉토리(src/main/java)에 서비스 클래스 생성(Calculator)

```
import org.springframework.stereotype.Service;
@Service
public class Calculator
{
    public int addition(int num1,int num2)
    {
        int R=num1+num2;
        return R;
    }
    public int subtraction(int num1,int num2)
    {
        int Res=num1-num2;
        return Res;
    }
    public int multiplication(int num1,int num2)
    {
        int Res=num1*num2;
        return Res;
    }
}
```

Maven Project

- Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - 기본 디렉토리에 Controller 클래스 생성(CalculatorController)

```
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
public class CalculatorController {

    private final Calculator calculator;

    public CalculatorController(Calculator calculator) {
        this.calculator = calculator;
    }

    @RequestMapping("/")
    public String index(){
        return "health check";
    }
```

Maven Project

- ❖ Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - ● 기본 디렉토리에 Controller 클래스 생성(CalculatorController)

```
    @RequestMapping("/add")
    public String add(@RequestParam("num1") int num1, @RequestParam("num2")
int num2){
        return String.valueOf(calculator.addition(num1,num2));
    }
    @RequestMapping("/sub")
    public String sub(@RequestParam("num1") int num1, @RequestParam("num2") int
num2){
        return String.valueOf(calculator.subtraction(num1,num2));
    }
    @RequestMapping("/mul")
    public String mul(@RequestParam("num1") int num1, @RequestParam("num2")
int num2){
        return String.valueOf(calculator.multiplication(num1,num2));
    }
}
```

Maven Project

- Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - tests 디렉토리에 Test 클래스 생성(TestAdditionFunctionality)

```
import org.testng.annotations.AfterClass;
import org.testng.annotations.BeforeClass;
import org.testng.annotations.BeforeGroups;
import org.testng.annotations.BeforeMethod;
import org.testng.annotations.Test;
import org.testng.Assert;

public class TestAdditionFunctionality
{

        Calculator Obj;
        int Result;

        @BeforeGroups("RegressionTest")
        public void InitGroup()
        {
                System.out.println("Im in Before Group");
                Obj=new Calculator();
        }
```

Maven Project

- Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - tests 디렉토리에 Test 클래스 생성(TestAdditionFunctionality)

```
@BeforeClass
public void Init()
{
        System.out.println("Im in Before class");
        Obj=new Calculator();
}

@BeforeMethod
public void ReinitialiseResultVar()
{
        System.out.println("I am in Before Method");
        Result=0;
}
```

Maven Project

- Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - tests 디렉토리에 Test 클래스 생성(TestAdditionFunctionality)

```
@Test(priority=1)
public void TestAdditionWithPositiveNumbers()
{
        System.out.println("I am in 1 st TestCase");
        Result=Obj.addition(10,20);
        Assert.assertEquals(Result, 30,"Addition does not work with
positive numbers");
}

@AfterClass
public void Teardown()
{
        System.out.println("I am in After Class");
        Obj=null;
}
}
```

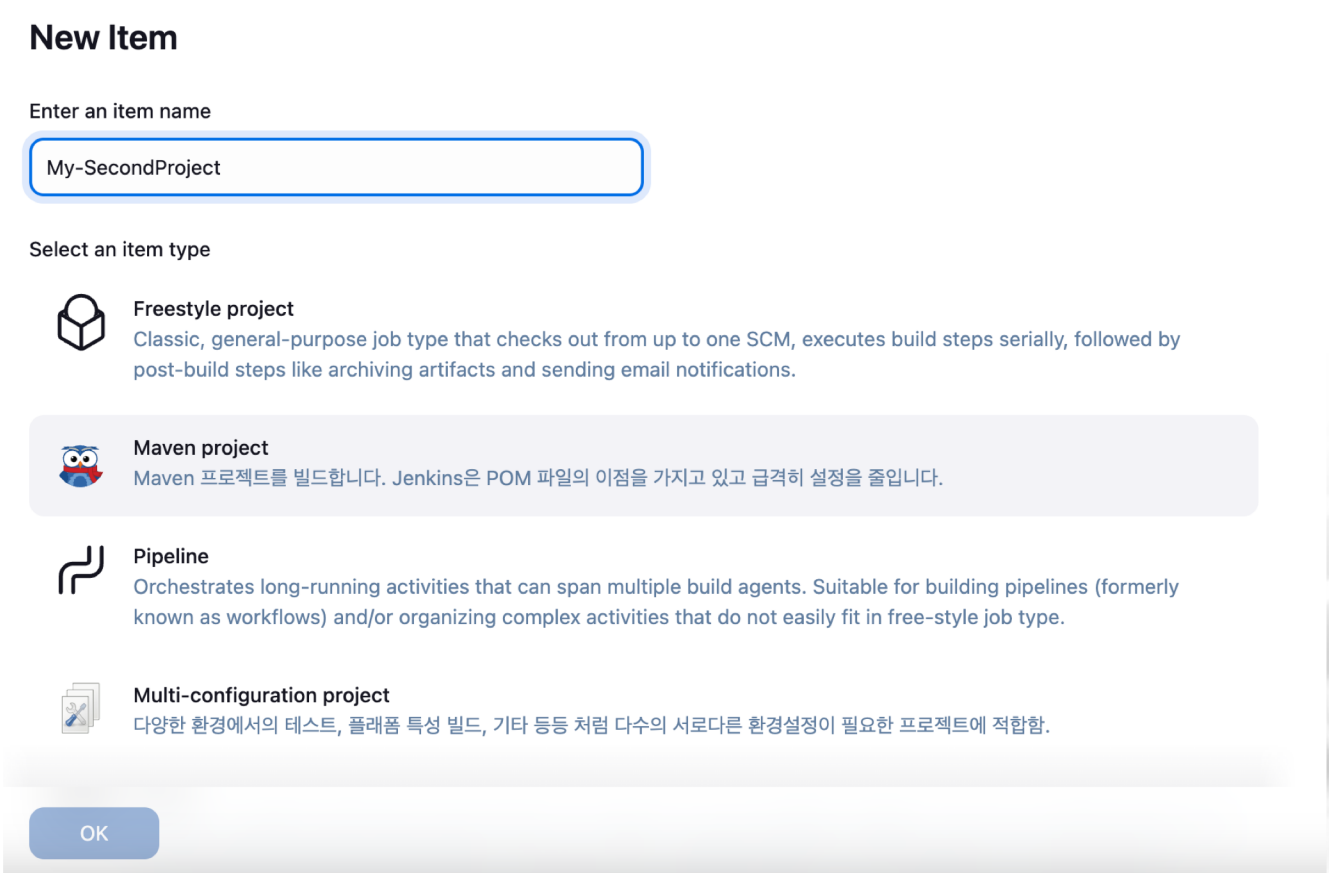
Maven Project

- Maven Project 생성 및 GitHub Push
  - ✓ Maven Project 생성
    - application.properties 에 서버 포트 설정

      server.port=80
    - 실행 한 후 테스트
  - ✓ Maven Project GitHub 리포지토리에 Push

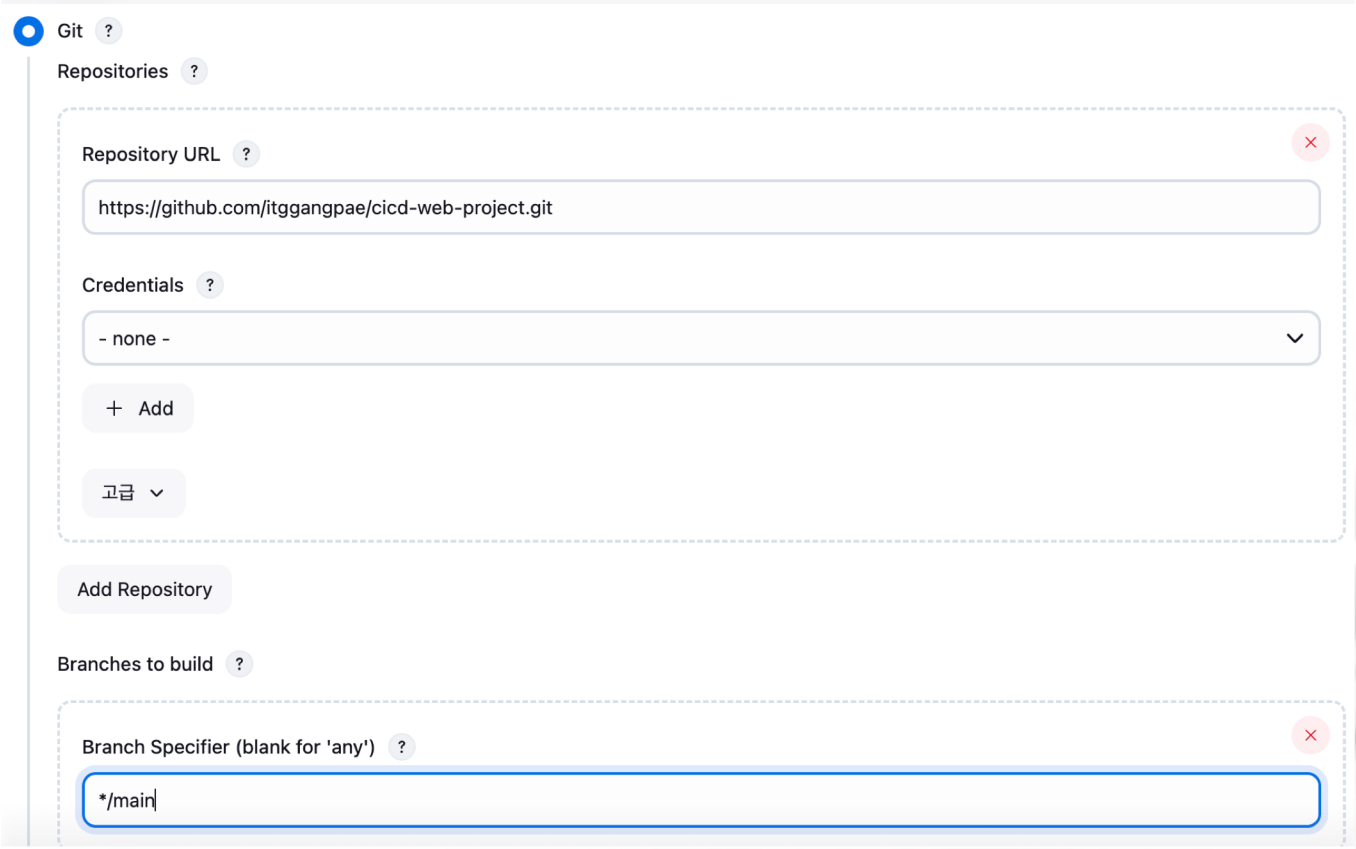
Maven Project

- Maven Project 빌드
  - ✓ Maven Project로 Item 생성



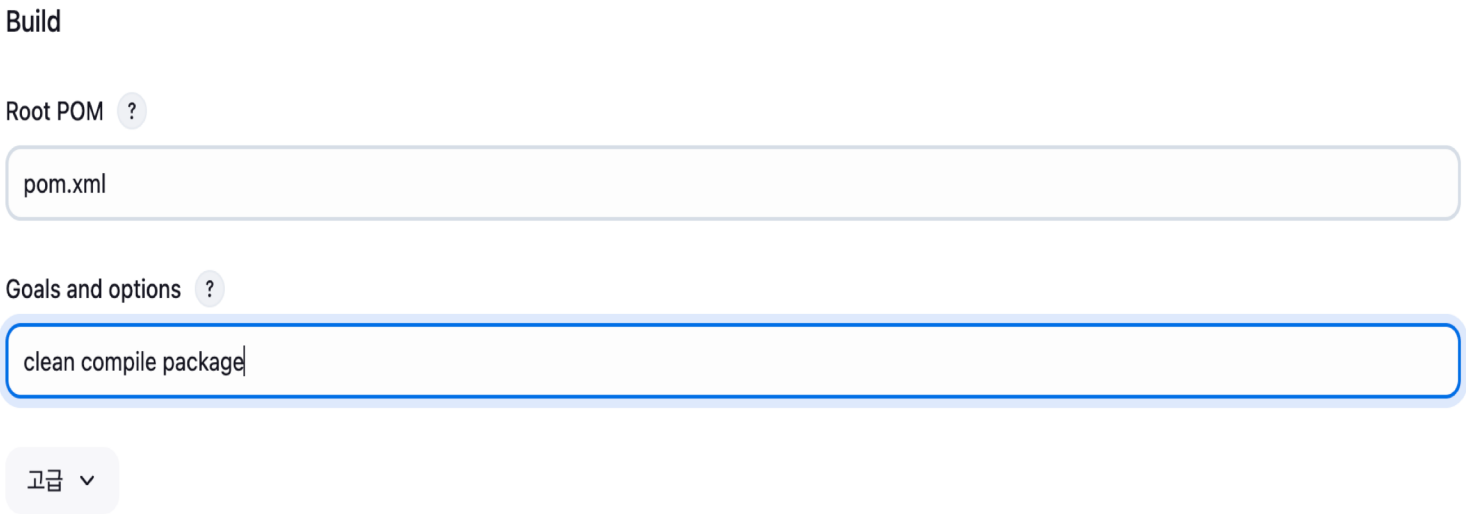
Maven Project

- ❖ Maven Project 빌드
  - ✓ 소스 코드 연결



Maven Project

- Maven Project 빌드
  - ✓ 빌드 명령어 설정



Maven Project

❖ Maven Project 빌드

✓ 빌드

```
[INFO] Packaging webapp
[INFO] Assembling webapp [web] in [/var/jenkins_home/workspace/My-SecondProject/target/hello-world]
[INFO] Processing war project
[INFO] Copying webapp resources [/var/jenkins_home/workspace/My-SecondProject/src/main/webapp]
[INFO] Webapp assembled in [17 msecs]
[INFO] Building war: /var/jenkins_home/workspace/My-SecondProject/target/hello-world.war
[INFO] ------------------------------------------------------------------------
[INFO] BUILD SUCCESS
[INFO] ------------------------------------------------------------------------
[INFO] Total time:  4.069 s
[INFO] Finished at: 2024-12-20T08:26:05Z
[INFO] ------------------------------------------------------------------------
Waiting for Jenkins to finish collecting data
[JENKINS] Archiving /var/jenkins_home/workspace/My-SecondProject/pom.xml to com.njonecompany.web/web/1.0/web-1.0.pom
[JENKINS] Archiving /var/jenkins_home/workspace/My-SecondProject/target/hello-world.war to com.njonecompany.web/web/1.0/web-
1.0.war
channel stopped
Finished: SUCCESS
```

Maven Project

- Maven Project 빌드
  - ✓ 결과 확인

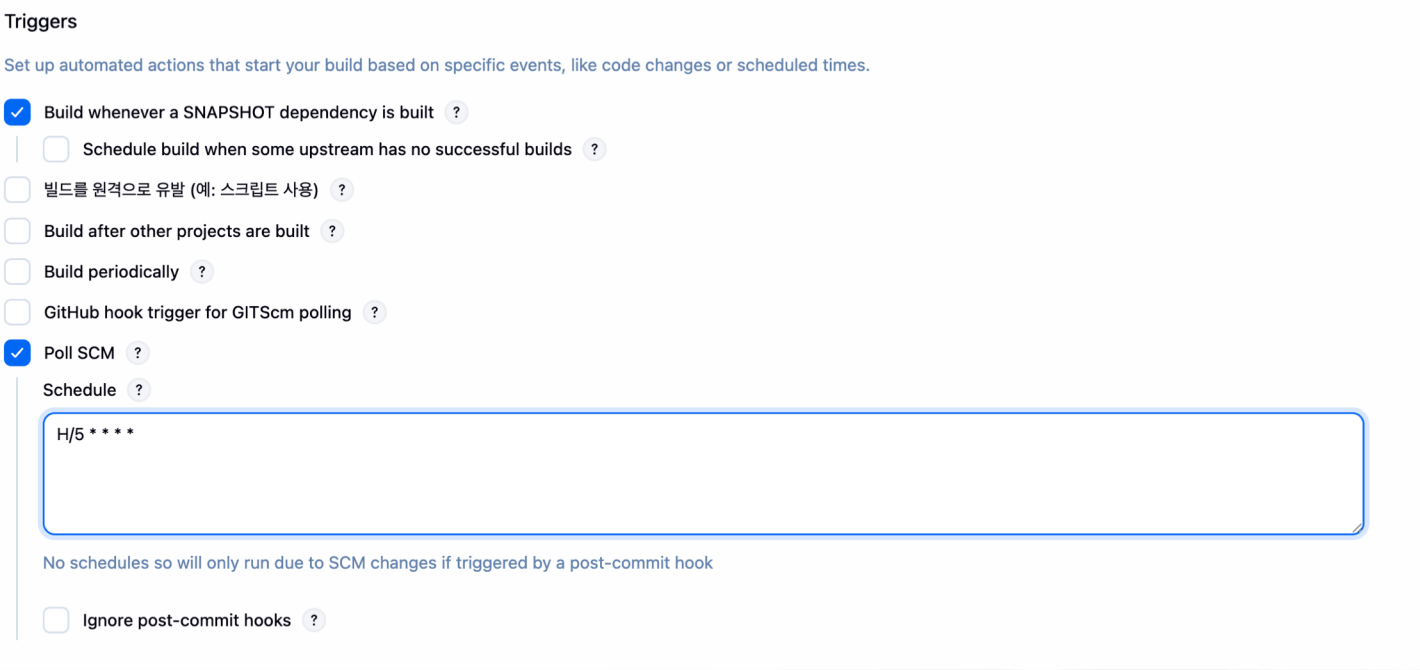
    docker exec -it jenkins-server /bin/bash

    ls /var/jenkins_home/workspace/프로젝트이름/target/

classes	hello-world	maven-status
generated-sources	hello-world.war	surefire-reports
generated-test-sources	maven-archiver	test-classes

Maven Project

- 일정 시간 단위로 소스 코드 레포지토리를 확인해서 업데이트 하기
  - ✓ 구성 수정

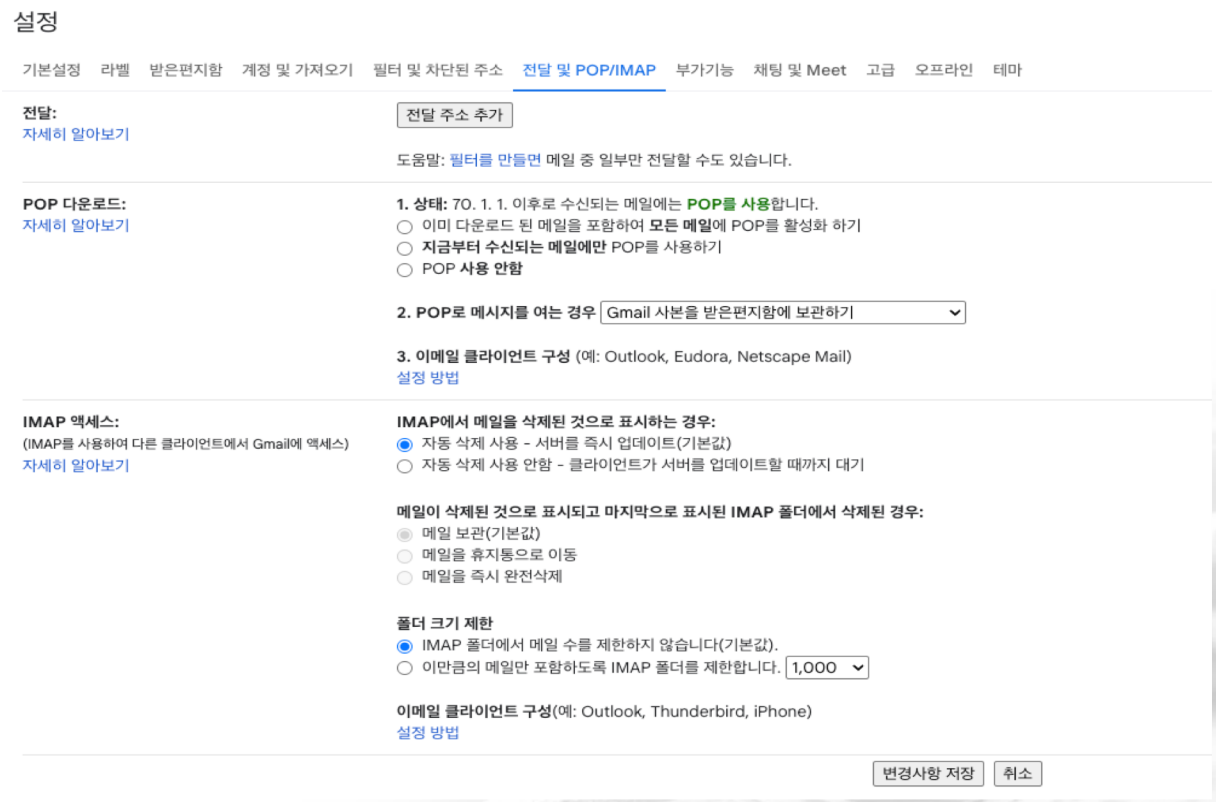


Maven Project

- ❖ 이메일 알림
  - ✓ 메일 서버 설정
    - 메일 서버가 있는 경우는 메일 서버 정보 확인
    - gmail 이나 naver 메일의 경우는 앱 비밀번호 생성

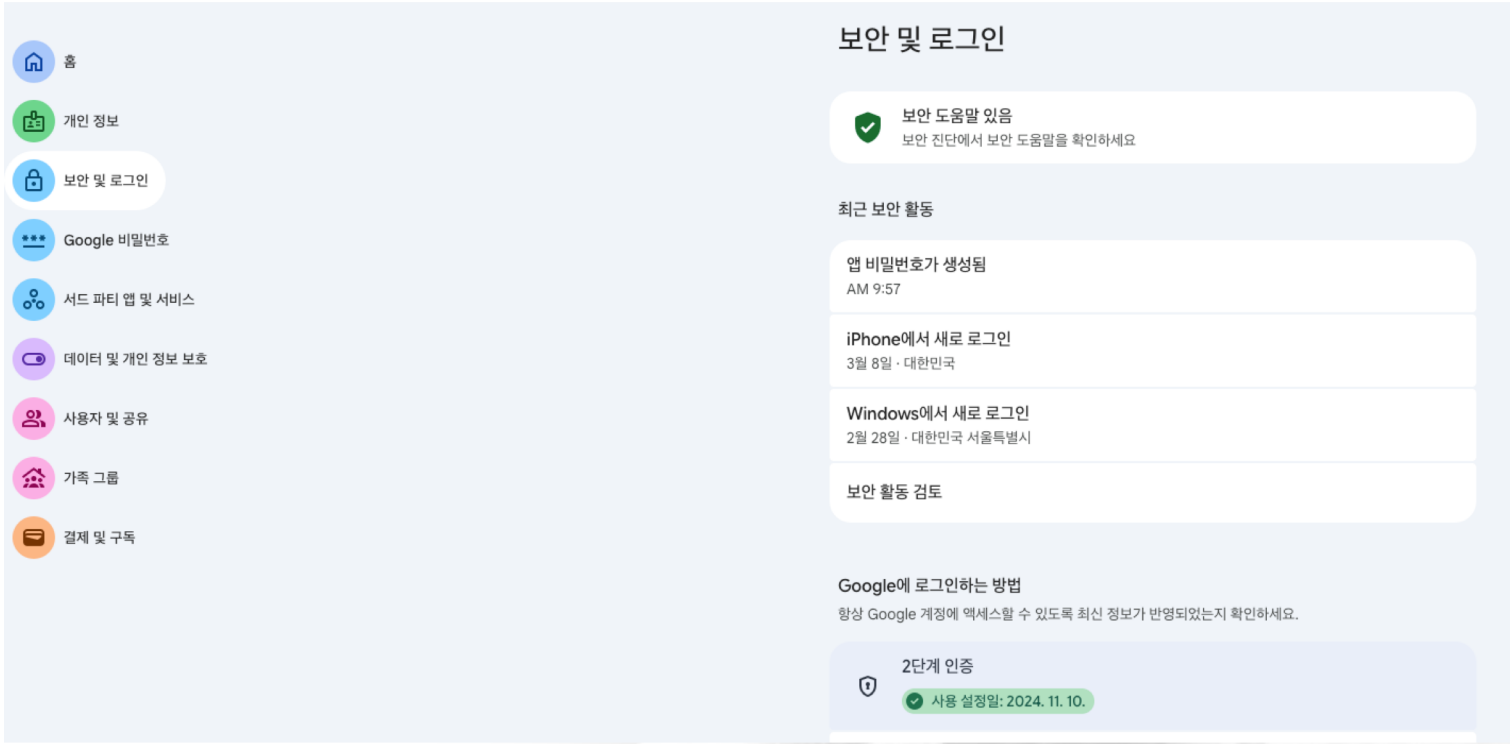
Maven Project

- ❖ 이메일 알림
  - ✓ 메일 서버 설정
    - Gmail 설정 – 메일 설정



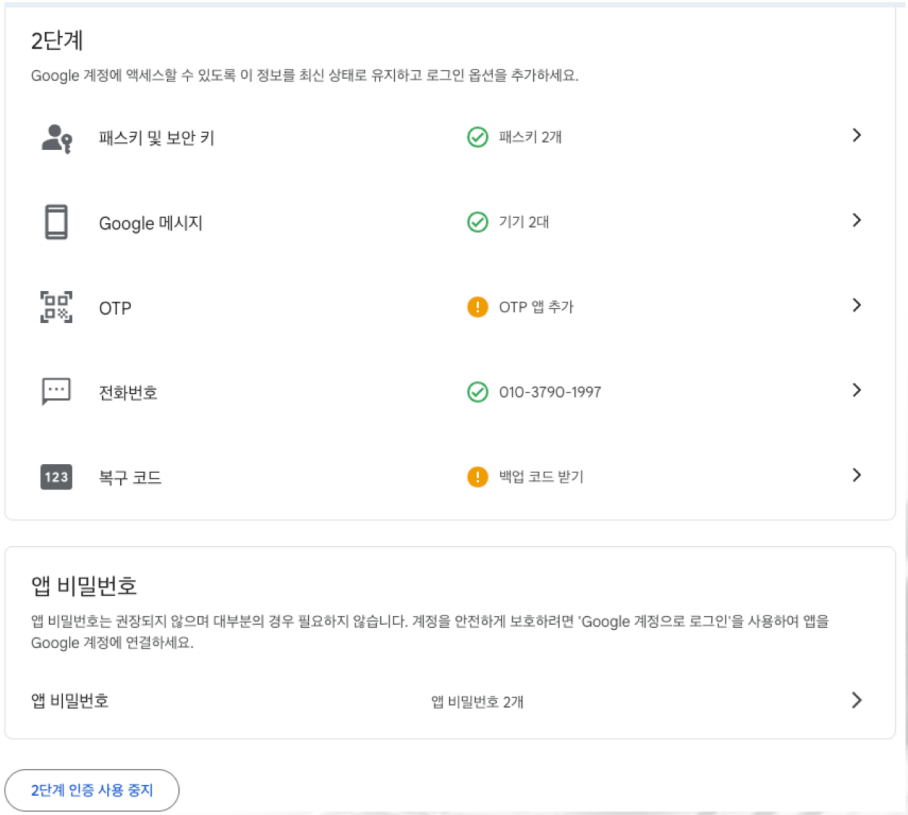
Maven Project

- ❖ 이메일 알림
  - ✓ 메일 서버 설정
    - Gmail 설정 – 앱 비밀번호 생성(계정 관리에서 수행 – 2단계 인증)



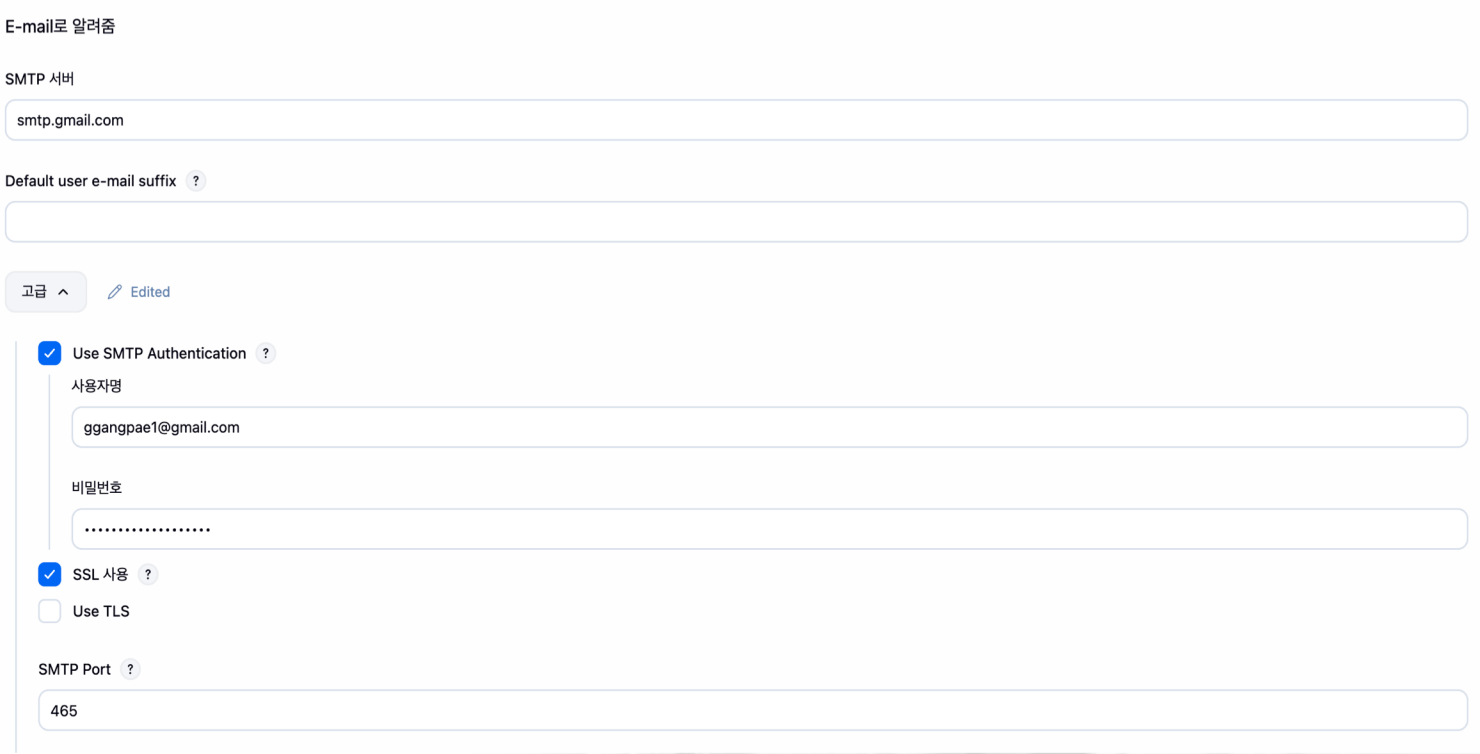
Maven Project

- ❖ 이메일 알림
  - ✓ 메일 서버 설정
    - Gmail 설정 – 앱 비밀번호 생성(계정 관리에서 수행 – 앱 비밀번호 생성)



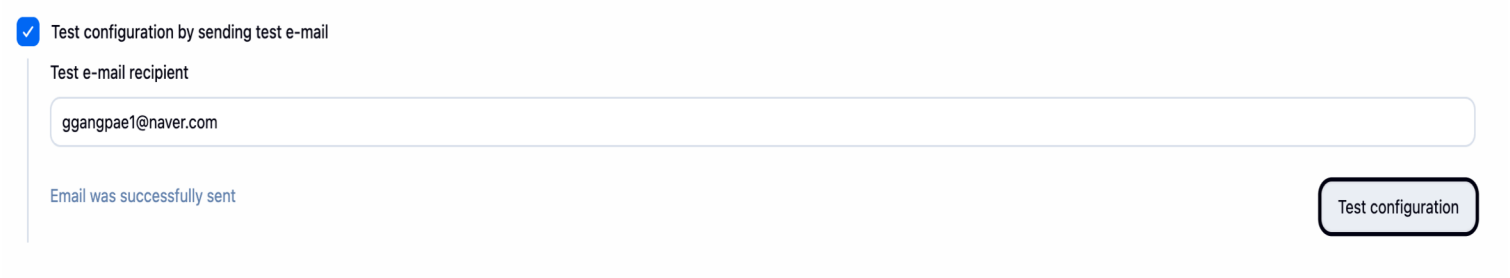
Maven Project

- ❖ 이메일 알림
  - ✓ 젠킨스 설정
    - ● 메일 서버 정보 설정



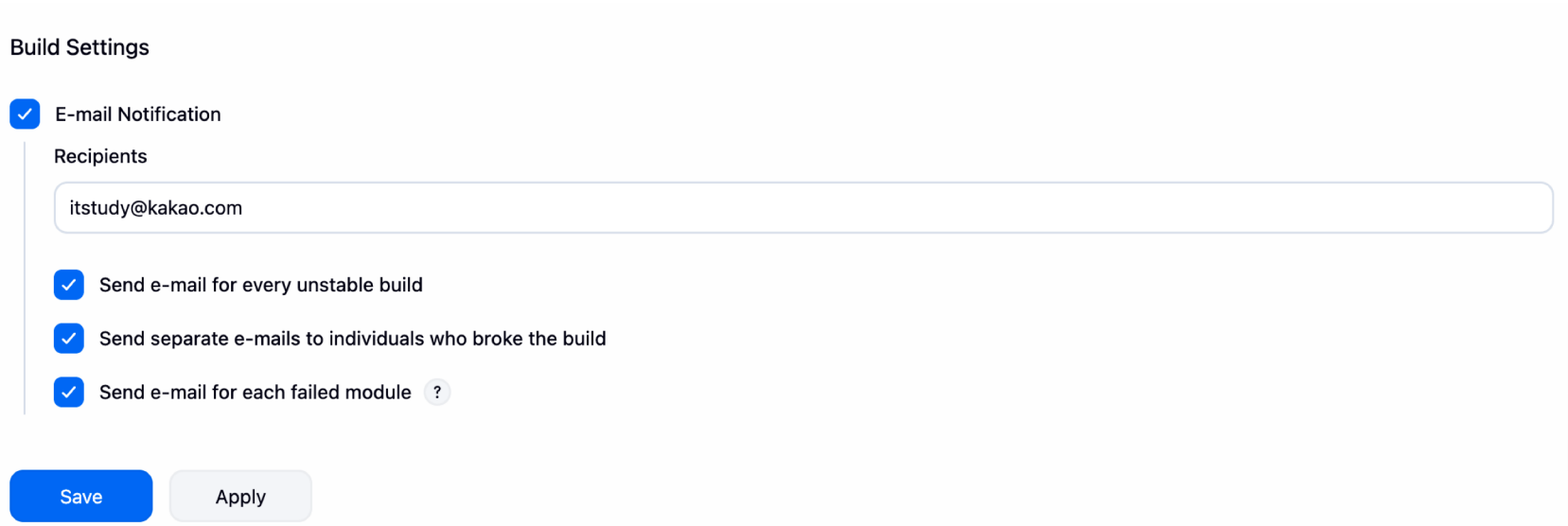
Maven Project

- ❖ 이메일 알림
  - ✓ 젠킨스 설정
    - ● 메일 전송 테스트



Maven Project

- ❖ 이메일 알림
  - ✓ 메일 서버 알림
    - ● 설정



SSH를 이용한 배포

- Gradle Project 생성 및 실행
  - ✓ Spring DevTools, Lombok, Spring Web 라이브러리의 의존성을 가진 프로젝트 생성

  - ✓ Service 계층의 클래스 생성 및 작성(Calculator)

```
import org.springframework.stereotype.Service;

@Service
public class Calculator {
    public int sum(int a, int b) {
        return a + b;
    }
}
```

SSH를 이용한 배포

- Gradle Project 생성 및 실행
  - ✓ Controller 계층의 클래스 생성 및 작성(CalculatorController)

```
import lombok.RequiredArgsConstructor;
import org.springframework.web.bind.annotation.RequestMapping;
import org.springframework.web.bind.annotation.RequestParam;
import org.springframework.web.bind.annotation.RestController;

@RestController
@RequiredArgsConstructor
class CalculatorController {
   private final Calculator calculator;

    @RequestMapping("/")
    String health() {
        return "healthy";
    }

    @RequestMapping("/sum")
    String sum(@RequestParam("a") Integer a,
               @RequestParam("b") Integer b) {
        return String.valueOf(calculator.sum(a, b));
    }
}
```

SSH를 이용한 배포

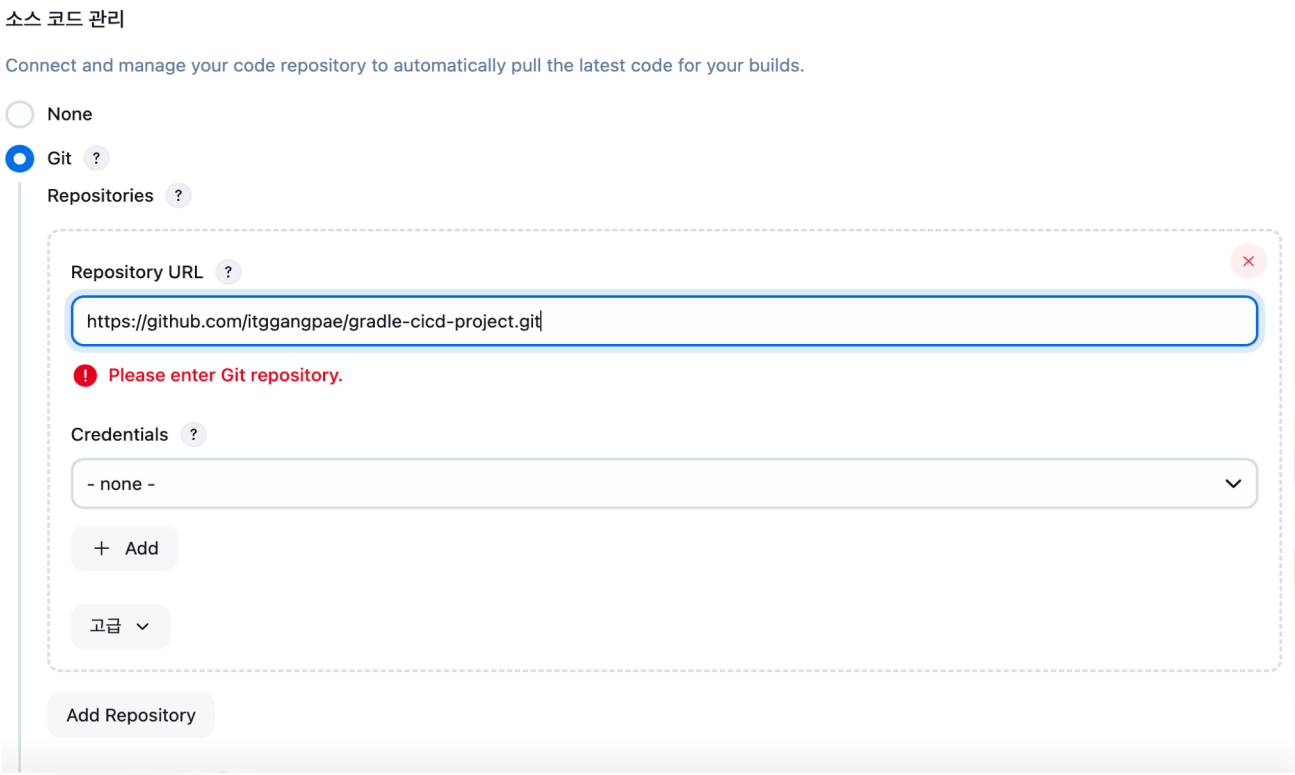
- Gradle Project 생성 및 실행
  - ✓ application.properties 파일에서 포트번호 변경
    server.port=9000
  - ✓ 프로젝트 컴파일: ./gradlew compileJava
  - ✓ 프로젝트 빌드: ./gradlew build
  - ✓ 프로젝트 패키징: ./gradlew jar
  - ✓ 프로젝트 실행: ./gradlew bootRun

SSH를 이용한 배포

- GitHub에 push

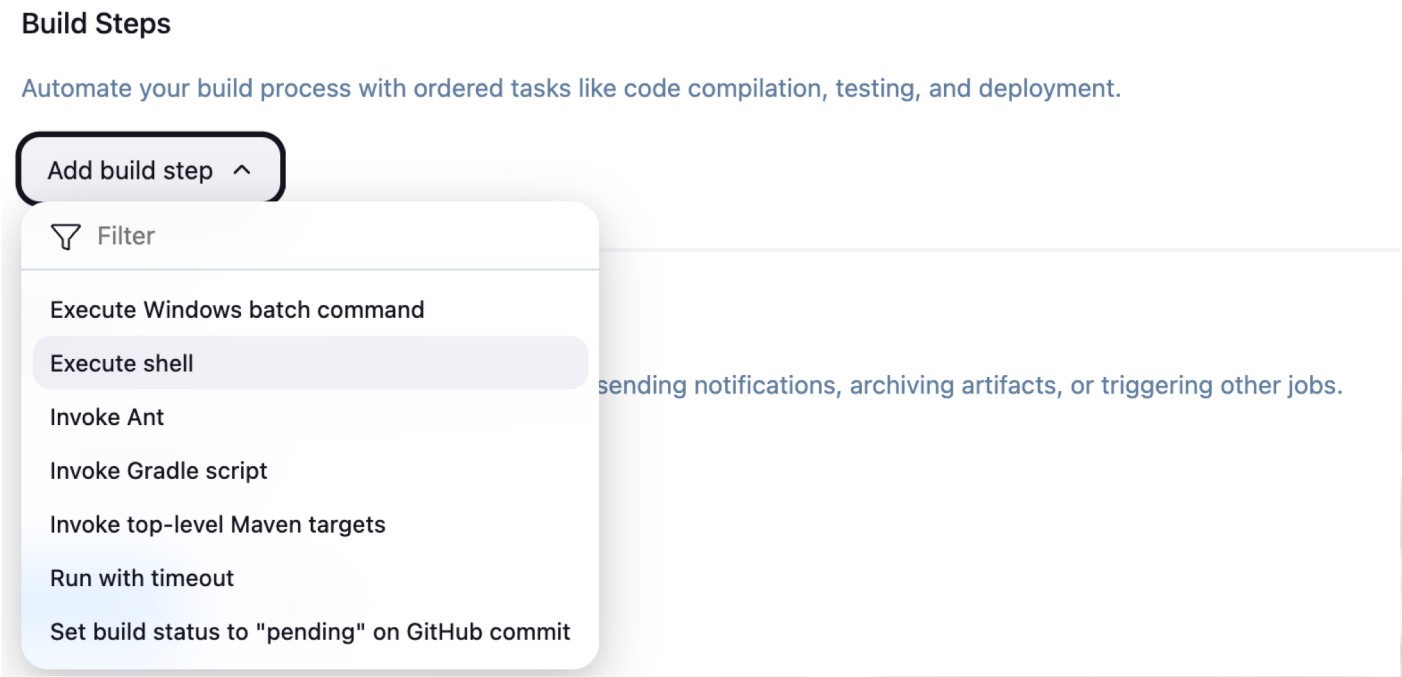
SSH를 이용한 배포

- ❖ Jenkins를 이용한 Gradle 빌드
  - ✓ Jenkins를 이용한 Gradle 빌드
    - Jenkins 프로젝트 생성 – Freestyle Project
    - 소스 코드 연결



SSH를 이용한 배포

- ❖ Jenkins를 이용한 Gradle 빌드
  - ✓ Jenkins를 이용한 Gradle 빌드
    - Shell 명령어 작성



SSH를 이용한 배포

- ❖ Jenkins를 이용한 Gradle 빌드
  - ✓ Jenkins를 이용한 Gradle 빌드
    - Shell 명령어 작성

      chmod +x ./gradlew

      ./gradlew compileJava

      ./gradlew clean build
    - Jenkins 프로젝트 빌드

SSH를 이용한 배포

- ❖ Jenkins를 이용한 Gradle 빌드
  - ✓ Jenkins를 이용한 Gradle 빌드
    - 에러 발생: JDK를 찾지 못해서 발생하는 에러

      FAILURE: Build failed with an exception.

      * What went wrong:

      Could not determine the dependencies of task ':compileJava'.

      > Could not resolve all dependencies for configuration ':compileClasspath'.

        > Failed to calculate the value of task ':compileJava' property 'javaCompiler'.

          > Cannot find a Java installation on your machine (Linux 6.10.14-linuxkit aarch64) matching: {languageVersion=17, vendor=any vendor, implementation=vendor-specific, nativeImageCapable=false}. Toolchain download repositories have not been configured.
    - Gradle이 필요한 Java 버전을 알아서 내려받도록 설정할 수 있는데 프로젝트의 settings.gradle 파일 상단에 추가

      ```
      plugins {
          id 'org.gradle.toolchains.foojay-resolver-convention' version '0.10.0'
      }
      ```

SSH를 이용한 배포

- ❖ Jenkins를 이용한 Gradle 빌드
  - ✓ Jenkins를 이용한 Gradle 빌드
    - 에러 발생: 특정 버전을 사용할려고 하는 경우 에러 발생

      * What went wrong:

      Class org.gradle.jvm.toolchain.JvmVendorSpec does not have member field 'org.gradle.jvm.toolchain.JvmVendorSpec IBM_SEMERU '
    - 프로젝트의 gradle/wrapper/gradle-wrapper.properties 파일의 url을 수정

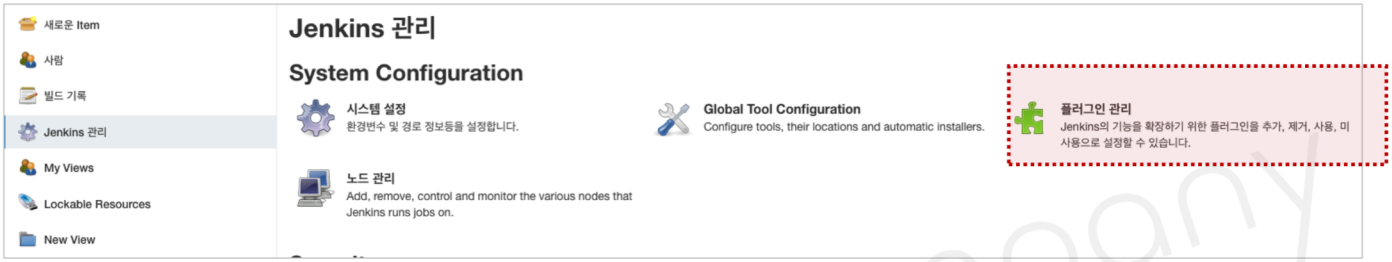
      distributionUrl=https₩://services.gradle.org/distributions/gradle-8.14-bin.zip

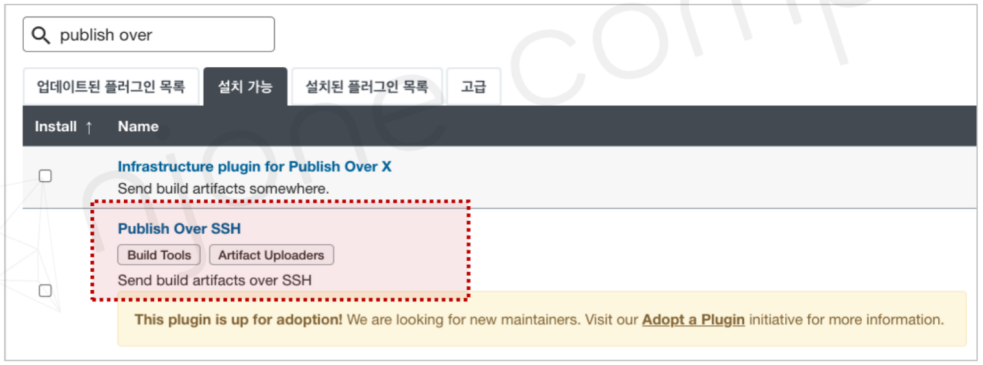
SSH를 이용한 배포

- SSH를 이용한 배포
  - ✓ SSH Server
    - SSH 서비스가 구동 중 인 서버 준비
    - SSH 서버에 JDK 설치: sudo apt install openjdk-17-jdk
    - SSH 서버의 home 디렉토리에 deploy 디렉토리 생성: mkdir ~/deploy

SSH를 이용한 배포

- ❖ SSH를 이용한 배포
  - ✓ Jenkins Server
    - ● publish Over SSH 플러그인 설치





SSH를 이용한 배포

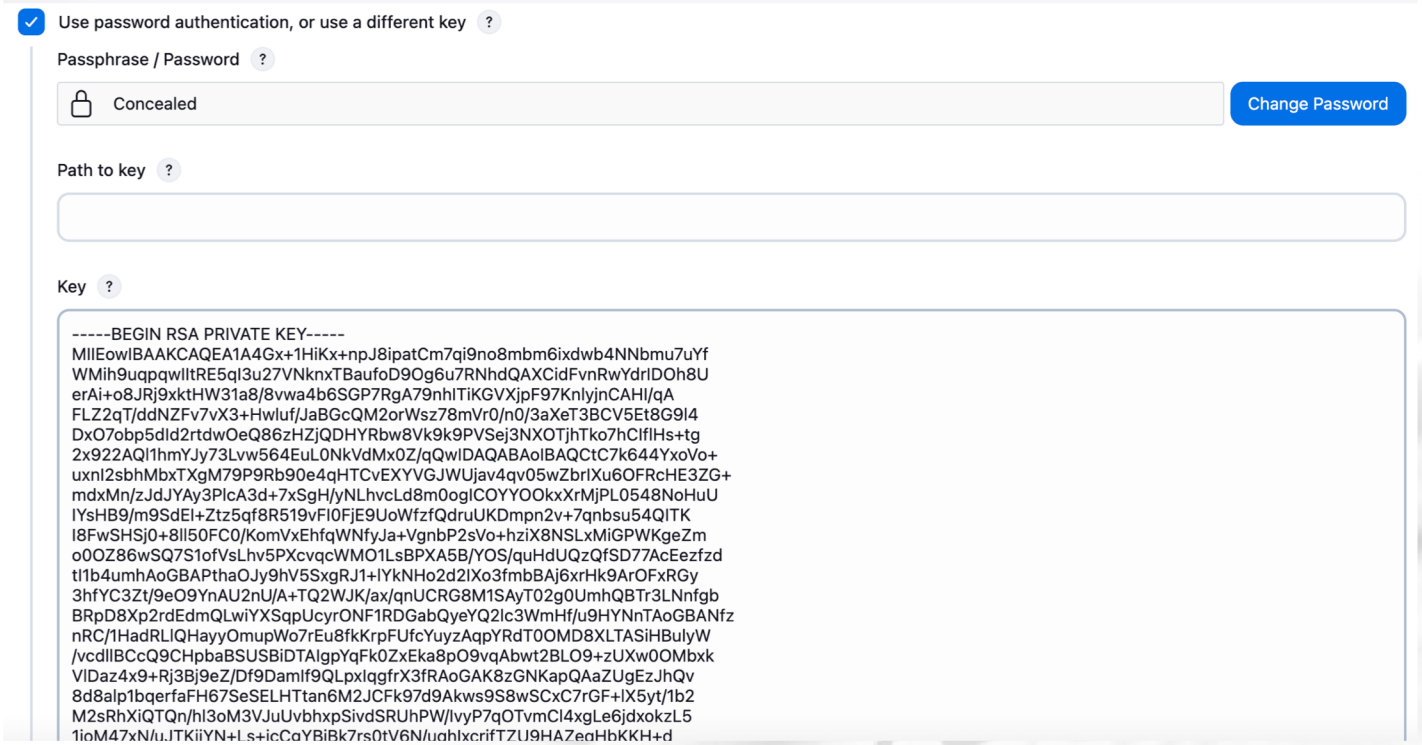
- SSH를 이용한 배포
  - ✓ Jenkins Server
    - SSH 접속 키 생성(EC2의 경우 pem 키값): ssh-keygen -t rsa -b 4096(Docker에 설치한 경우라면 docker exec -it <컨테이너_ID> /bin/bash 수행 후)
    - 생성된 id_rsa.pub 내용을 복사해서 배포 대상 서버(Remote Server)의 ~/.ssh/authorized_keys에 추가

SSH를 이용한 배포

- SSH를 이용한 배포
  - ✓ Jenkins Server
    - SSH Server 등록: Manage Jenkins > System > SSH Server

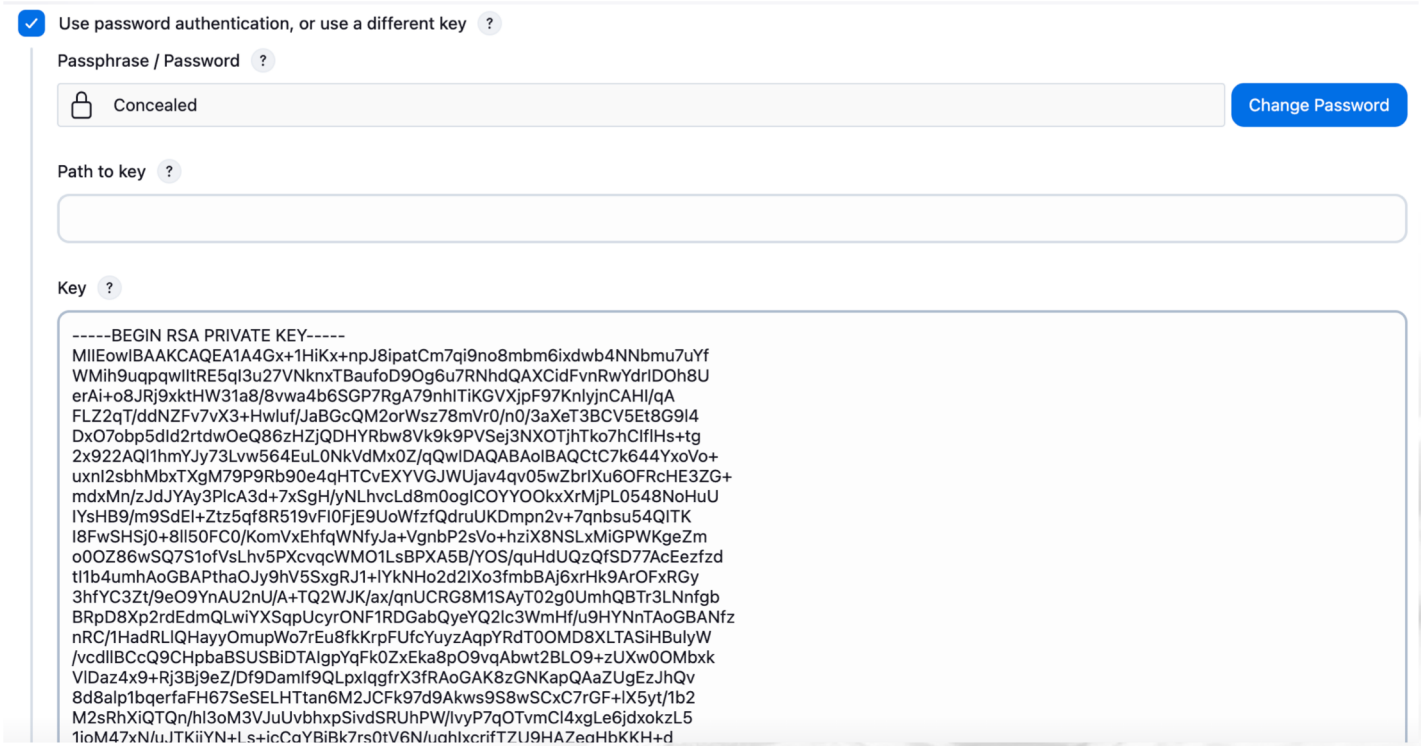
SSH를 이용한 배포

- SSH를 이용한 배포
  - ✓ Jenkins Server
    - SSH Server 등록: Manage Jenkins > System > SSH Server > 고급 > key 입력란에 컨테이너 내부의 비밀키 파일 내용을 통째로 붙여넣거나 해당 파일 경로를 입력



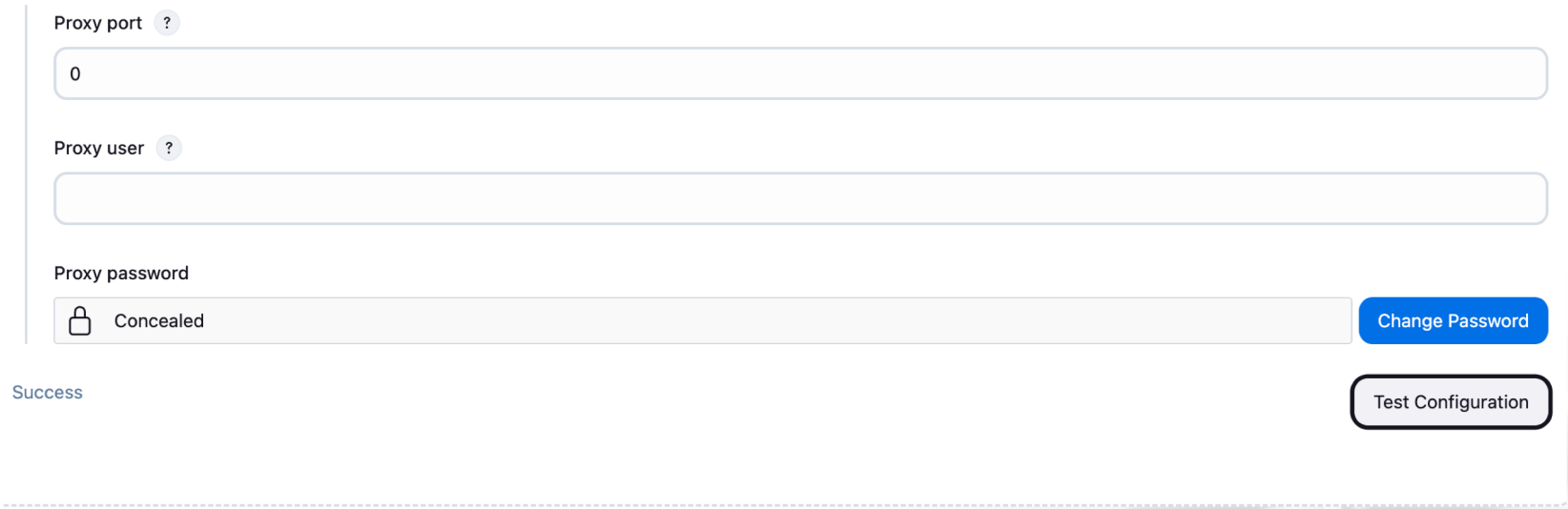
SSH를 이용한 배포

- SSH를 이용한 배포
  - ✓ Jenkins Server
    - SSH Server 등록: Manage Jenkins > System > SSH Server > 고급을 누르고 포트 번호를 설정



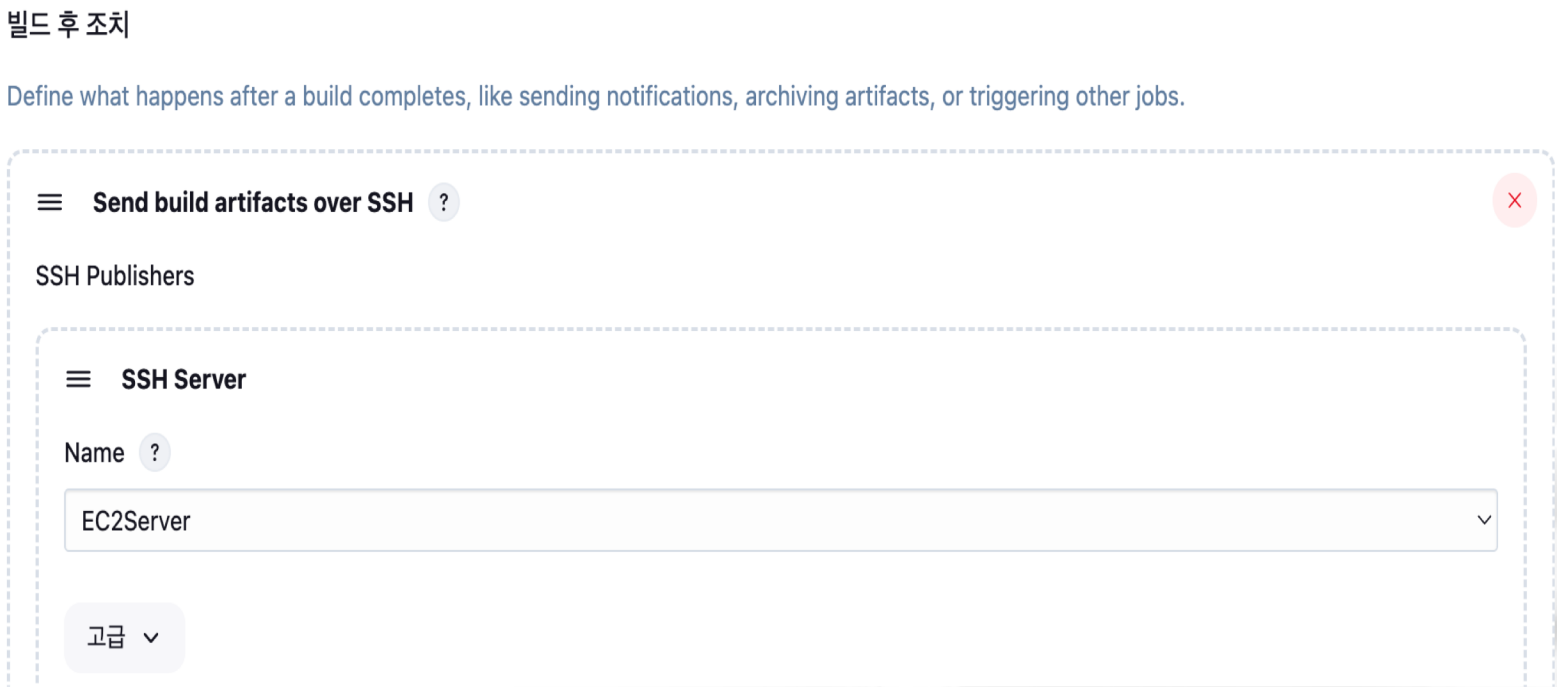
SSH를 이용한 배포

- SSH를 이용한 배포
  - ✓ Jenkins Server
    - SSH Server 등록: Manage Jenkins > System > SSH Server > Test Configuration



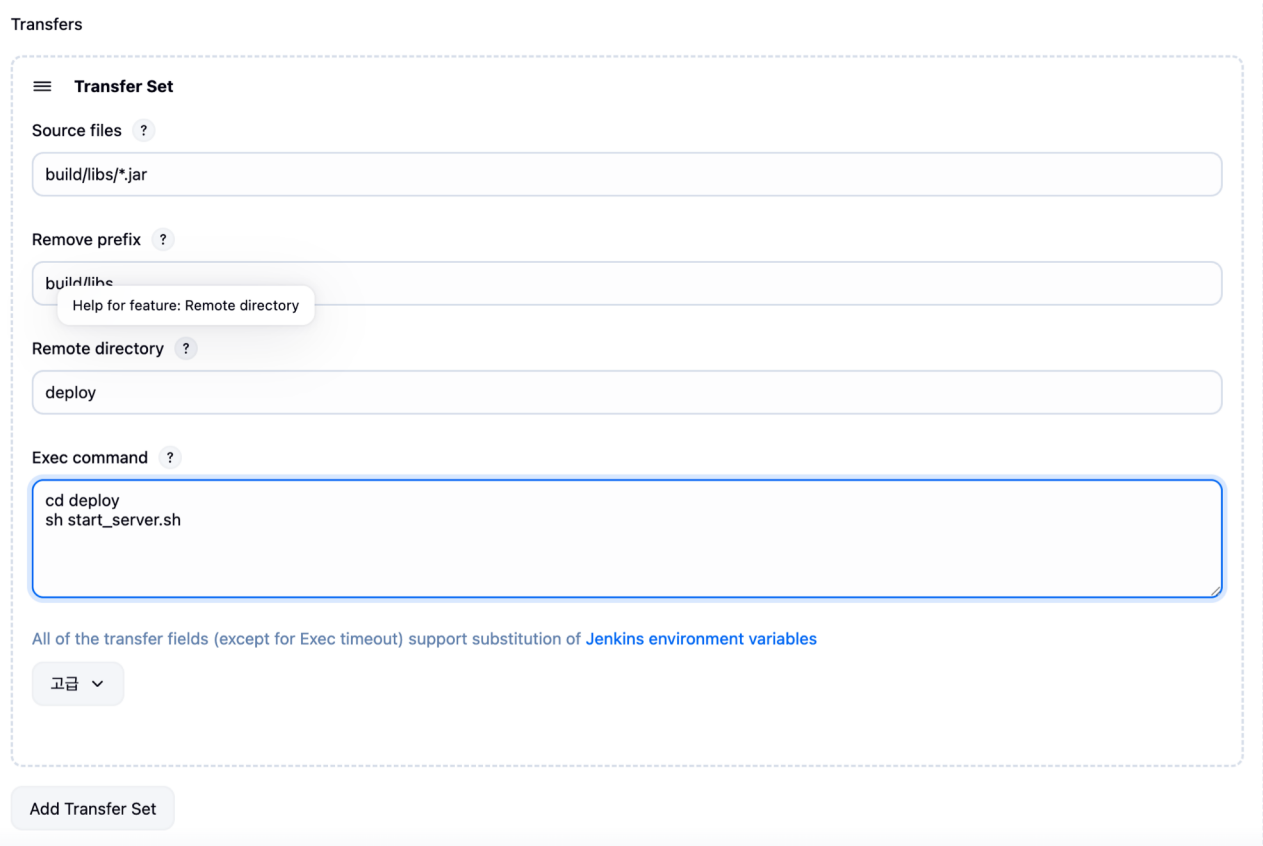
SSH를 이용한 배포

- SSH를 이용한 배포
  - ✓ Jenkins Server
    - 구성 > 빌드 후 조치



SSH 접속

- ❖ SSH를 이용한 배포
  - ✓ Jenkins Server
    - ● 구성 > 빌드 후 조치



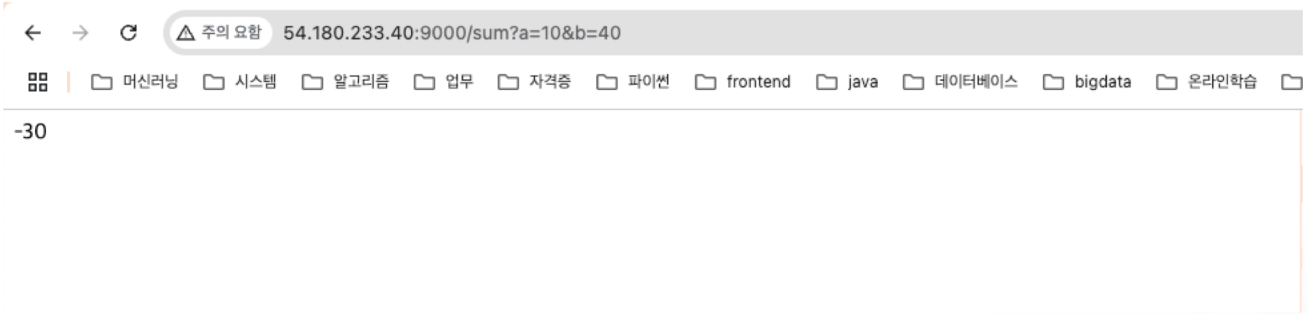
SSH 접속

- SSH를 이용한 배포
  - ✓ SSH_Server
    - /deploy 디렉토리에 server_start.sh 생성 및 작성

```
echo "Java Version" java -version

echo "Start Spring Boot Application!"
CURRENT_PID=$(ps -ef | grep java | grep calculator | awk '{print $2}')

echo $CURRENT_PID

if [ -z $CURRENT_PID ]; then
        echo ">현재 구동중인 어플리케이션이 없으므로 종료하지 않습니다."

else
        echo "> kill -15 $CURRENT_PID"
        kill -15 $CURRENT_PID
        sleep 10
fi

echo ">어플리케이션 배포 진행!"

nohup java -jar calculator-0.0.1-SNAPSHOT.jar
```

SSH 접속

- SSH를 이용한 배포
  - ✓ Jenkins_Server
    - 아이템 빌드
  - ✓ 브라우저에서 접속: http://IP:9000/sum?a=10&b=40

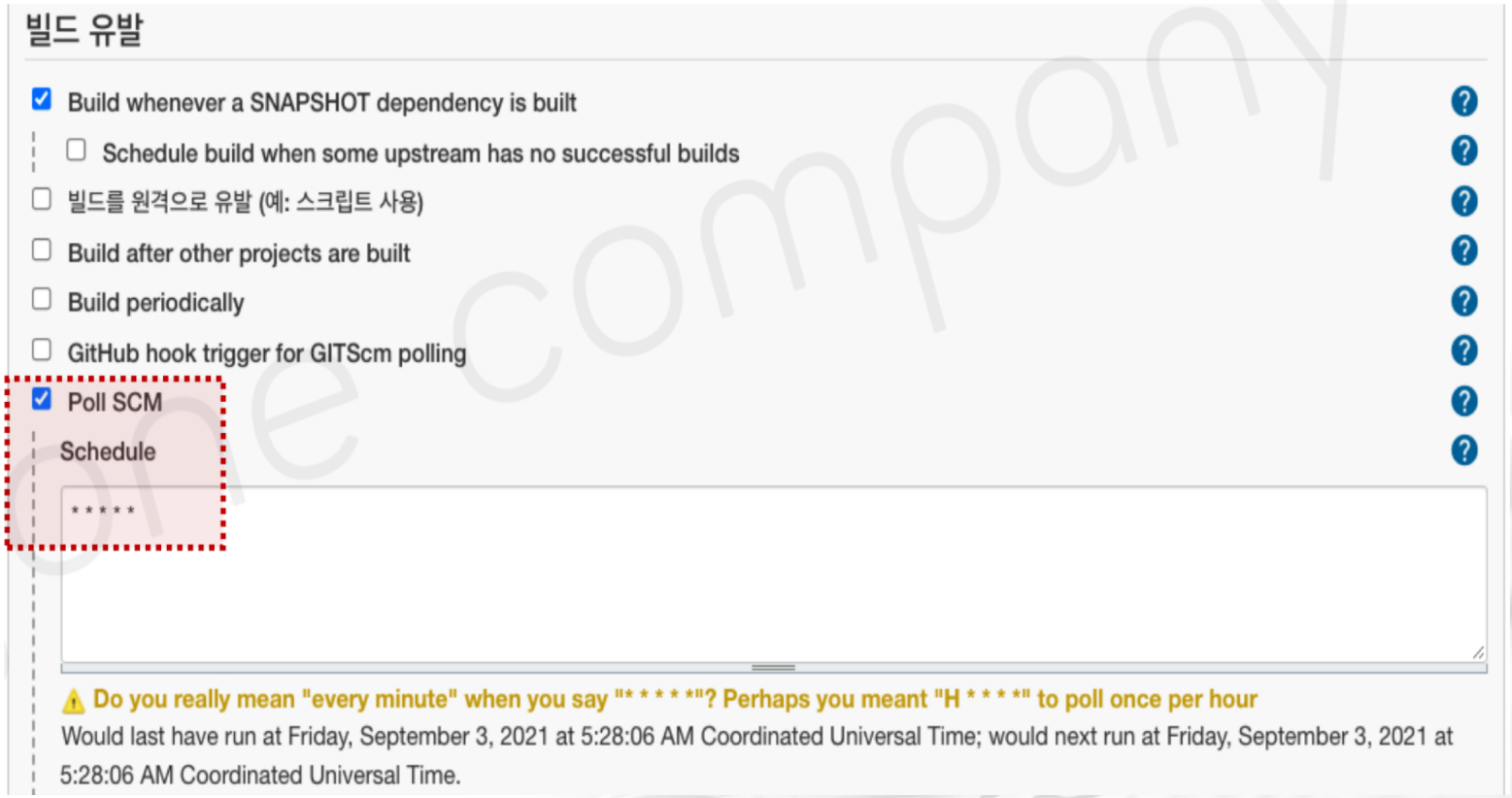


Triggers

- 주기적인 빌드
  - ✓ Cron Job
    - 소프트웨어 유틸리티 cron은 유닉스 계열 컴퓨터 운영체제의 시간 기반 잡 스케줄
    - 소프트웨어 환경을 설정하고 관리하는 사람들은 작업을 고정된 시간, 날짜, 간격에 주기적으로 실행할 수 있도록 스케줄링하기 위해 cron을 사용
  - ✓ Project > Configure > Triggers
    - Build periodically: 변경 내용이 없어도 주기적으로 빌드를 수행하는 cron job
    - Poll SCM: 변경 내역이 있는 경우에만 빌드를 수행하는 cron job

Triggers

- 주기적인 빌드
  - ✓ 실습 – Jenkins 가 Git에 접속해서 확인하는 방식
    - Trigger 작성



Triggers

- 주기적인 빌드
  - ✓ 실습
    - Calculator 클래스 소스 코드 수정

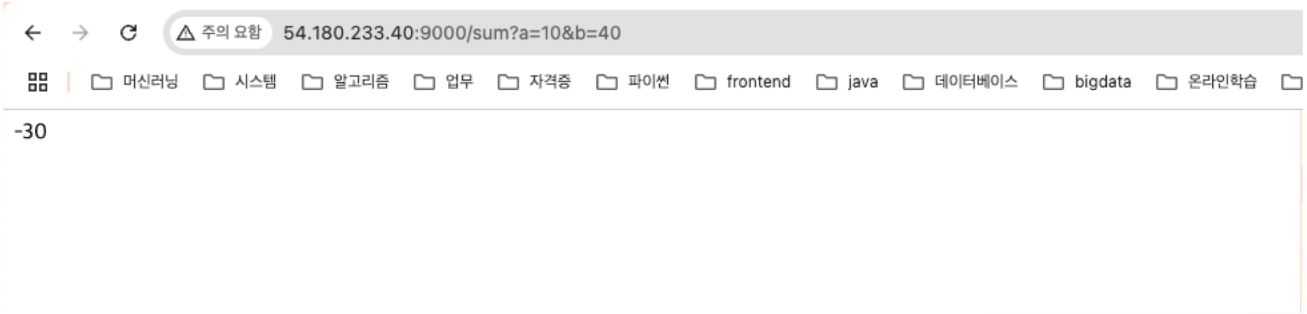
```
import org.springframework.stereotype.Service;

@Service
public class Calculator {
    public int sum(int a, int b) {
        return a * b;
    }
}
```

Triggers

- 주기적인 빌드
  - ✓ 실습
    - Git push
      - git add .
      - git commit –m “Poll SCM”
      - git push origin main

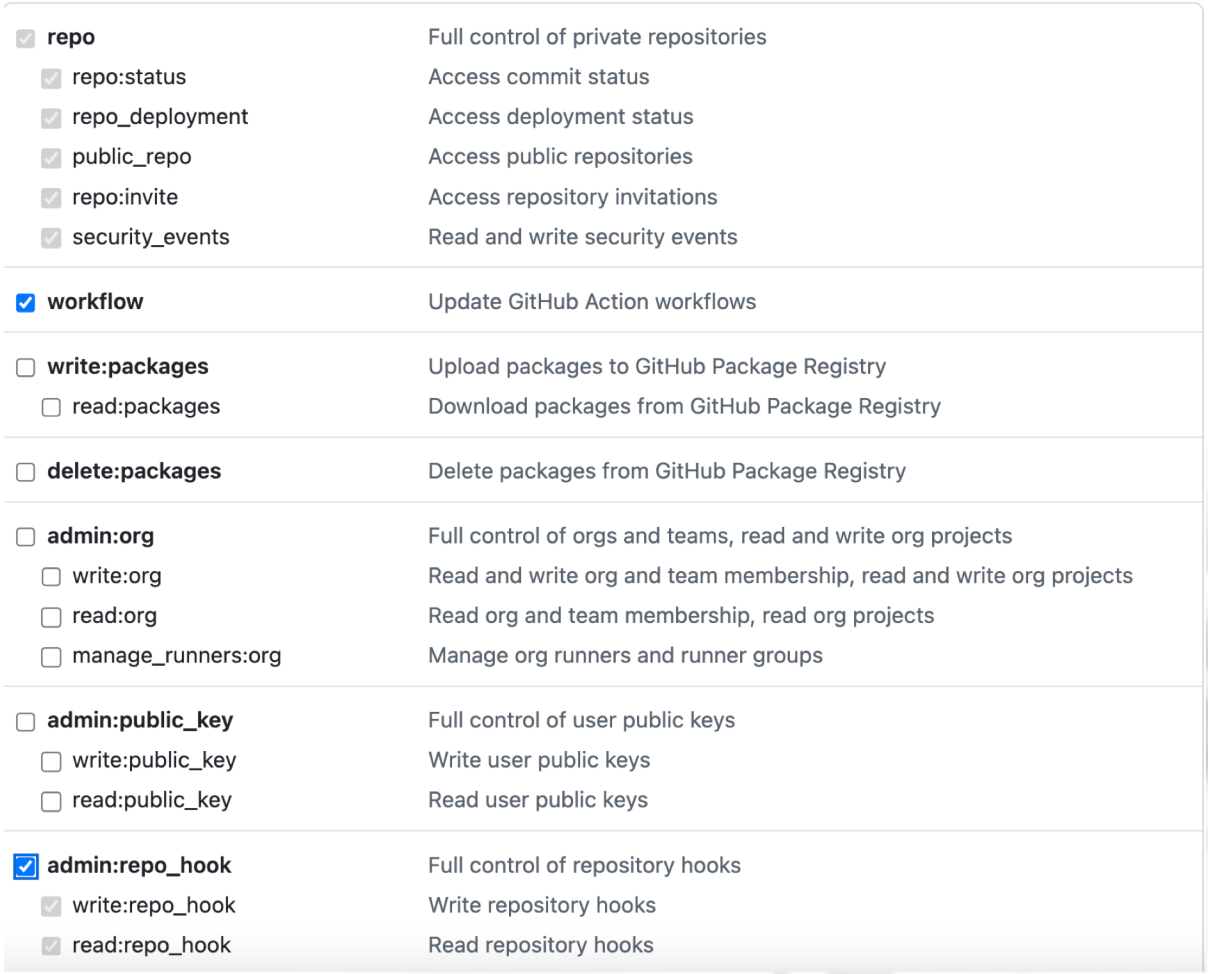
Triggers

- 주기적인 빌드
  - ✓ 실습
    - 브라우저에서 접속: http://IP:9000/sum?a=10&b=40



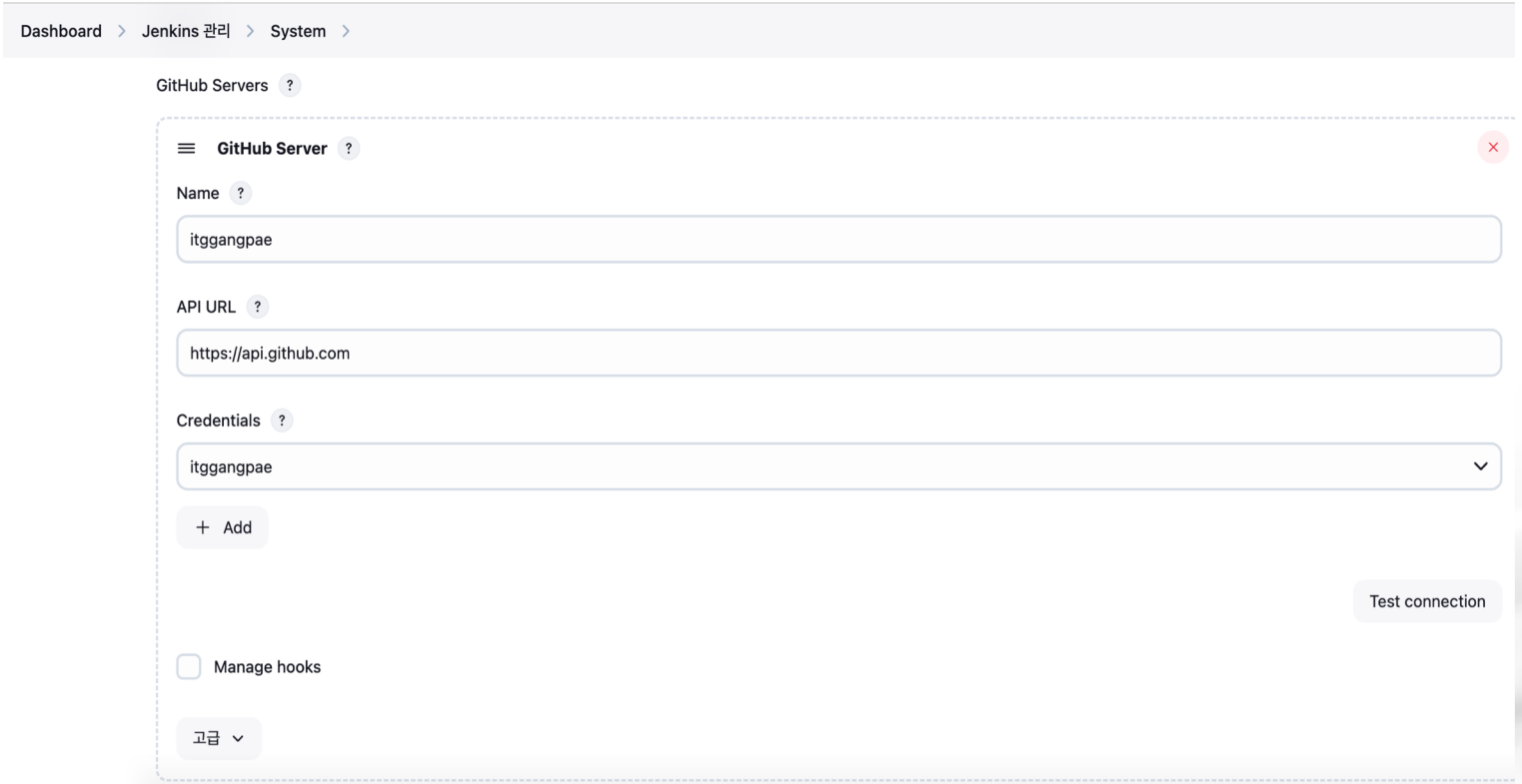
Triggers

- GitHub WebHook을 이용한 업데이트
  - ✓ GitHub에 접속해서 토큰 생성



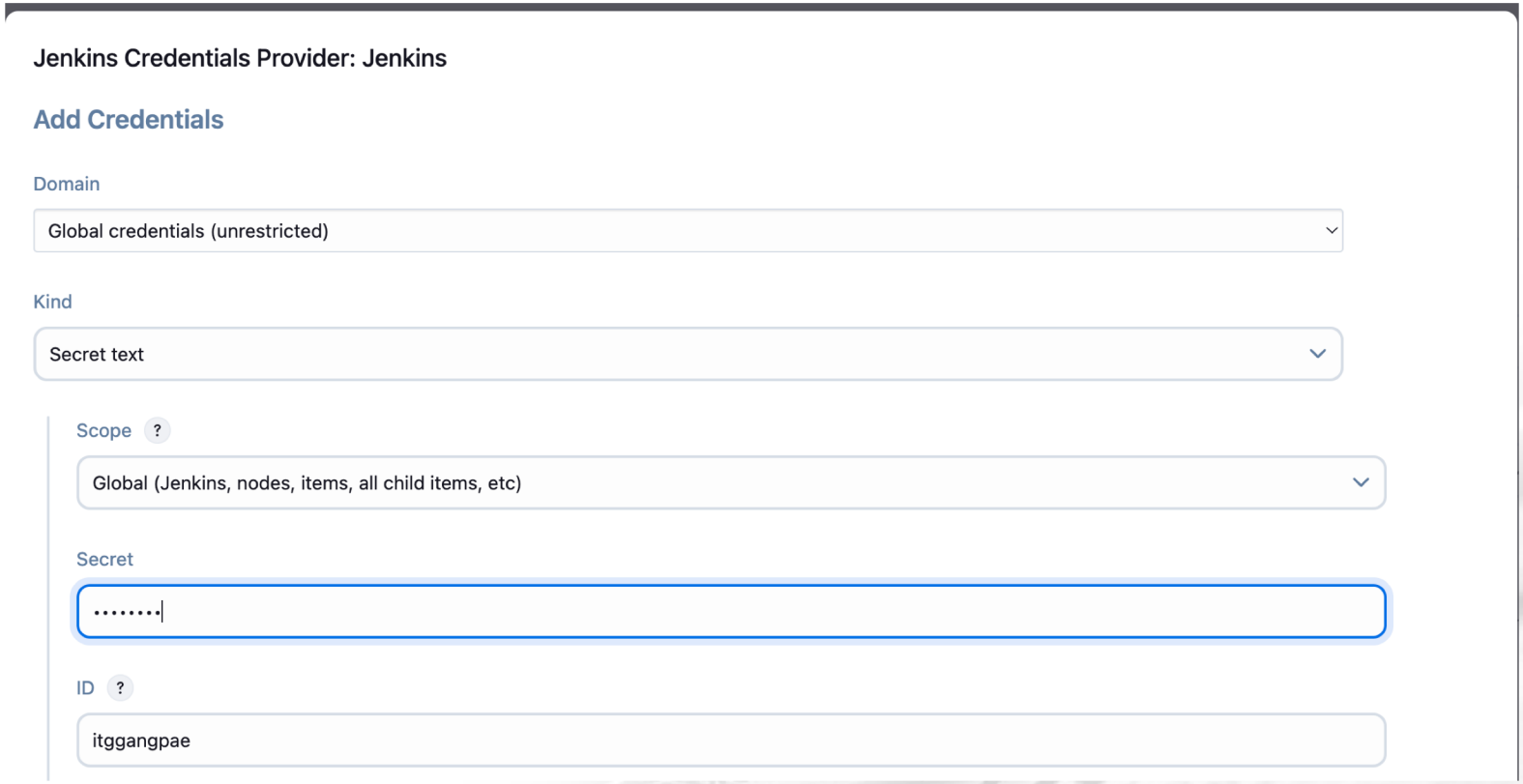
Triggers

- ❖ GitHub WebHook을 이용한 업데이트
  - ✓ Jenkins Server 설정



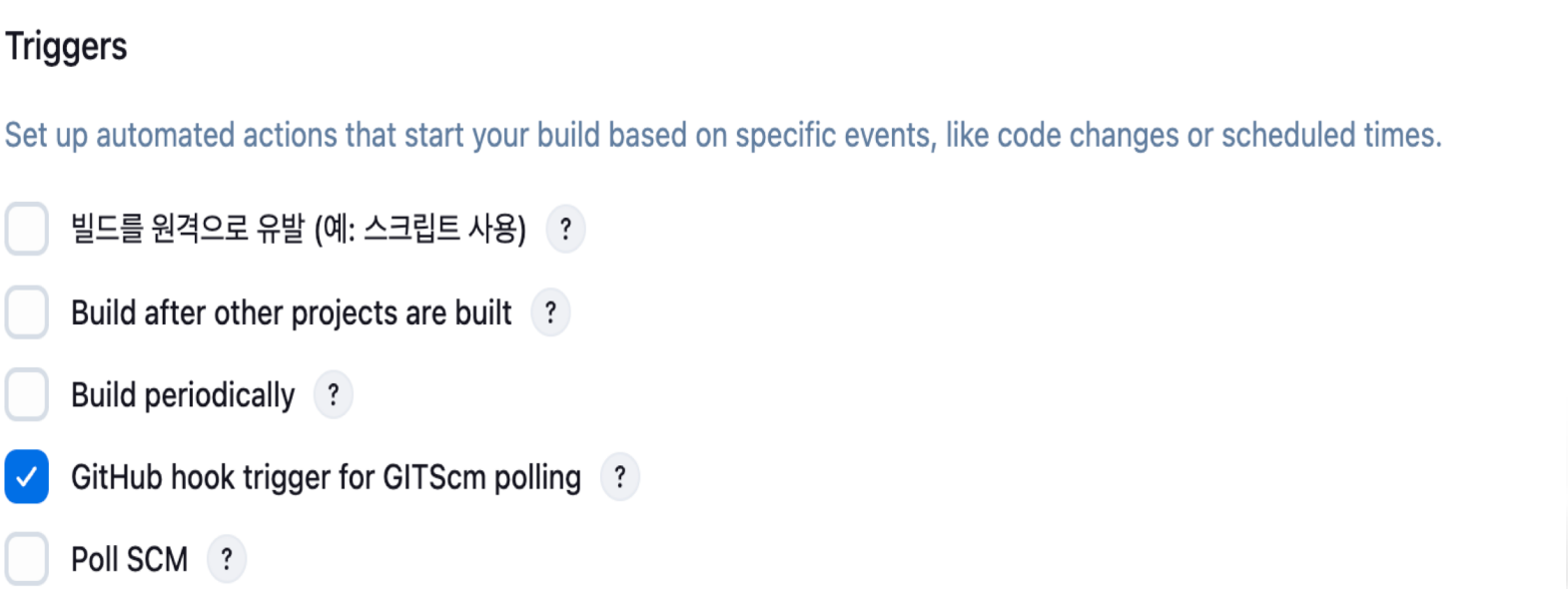
Triggers

- GitHub WebHook을 이용한 업데이트
  - ✓ Jenkins Server 설정
    - Credentials 밑에 +Add 를 선택한 후 인증정보를 기입



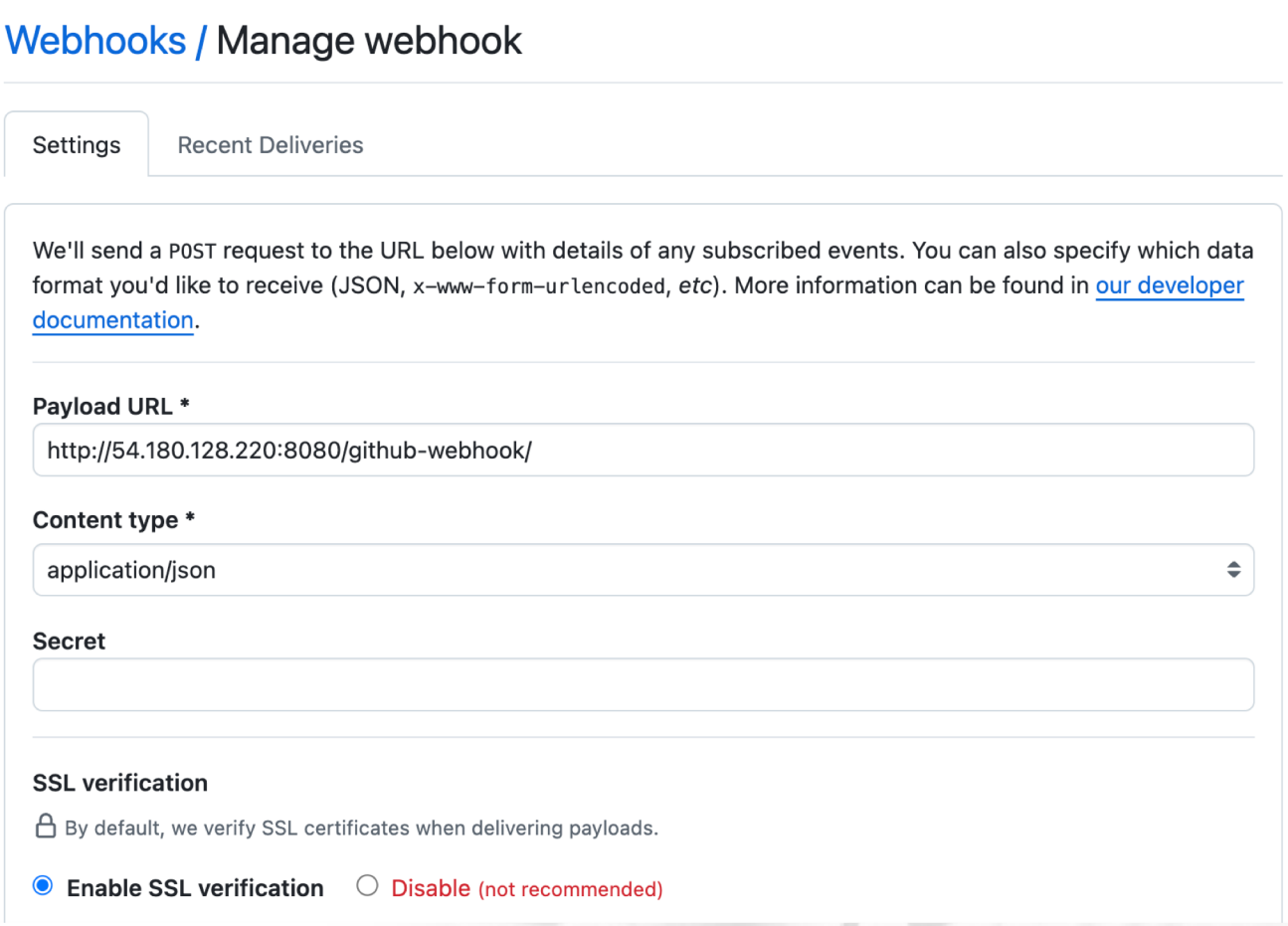
Triggers

- GitHub WebHook을 이용한 업데이트
  - ✓ Jenkins Item 설정



Triggers

- GitHub WebHook을 이용한 업데이트
  - ✓ GitHub 의 repository의 settings 탭에서 webhook 생성



Triggers

- GitHub WebHook을 이용한 업데이트
  - ✓ 코드 수정 후 push 한 후 확인

Docker Image 배포

- SSH를 이용한 도커 이미지 배포
  - ✓ SSH 서버의 deploy 디렉토리에 Dockerfile 생성

```
FROM amazoncorretto:17
COPY calculator-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

Docker Image 배포

- SSH를 이용한 도커 이미지 배포
  - ✓ Jenkins 서버의 빌드 후 조치의 exec command 수정

```
cd deploy
docker ps -q -f name=calculator | grep -q . && docker stop calculator
docker rmi -f $(docker images -q calculator) 2>/dev/null || echo "이미지 없음"
docker build --no-cache -t calculator -f Dockerfile .
docker run -d -p 8000:9000 --rm --name calculator calculator:latest
```

Docker Image 배포

- 도커 허브에 이미지 배포
  - ✓ 프로젝트에 Dockerfile 생성 후 작성

```
FROM amazoncorretto:17
CMD ["./mvnw", "clean", "package"]
COPY ./build/libs/calculator-0.0.1-SNAPSHOT.jar app.jar
ENTRYPOINT ["java", "-jar", "app.jar"]
```

  - ✓ GitHub에 코드 Push

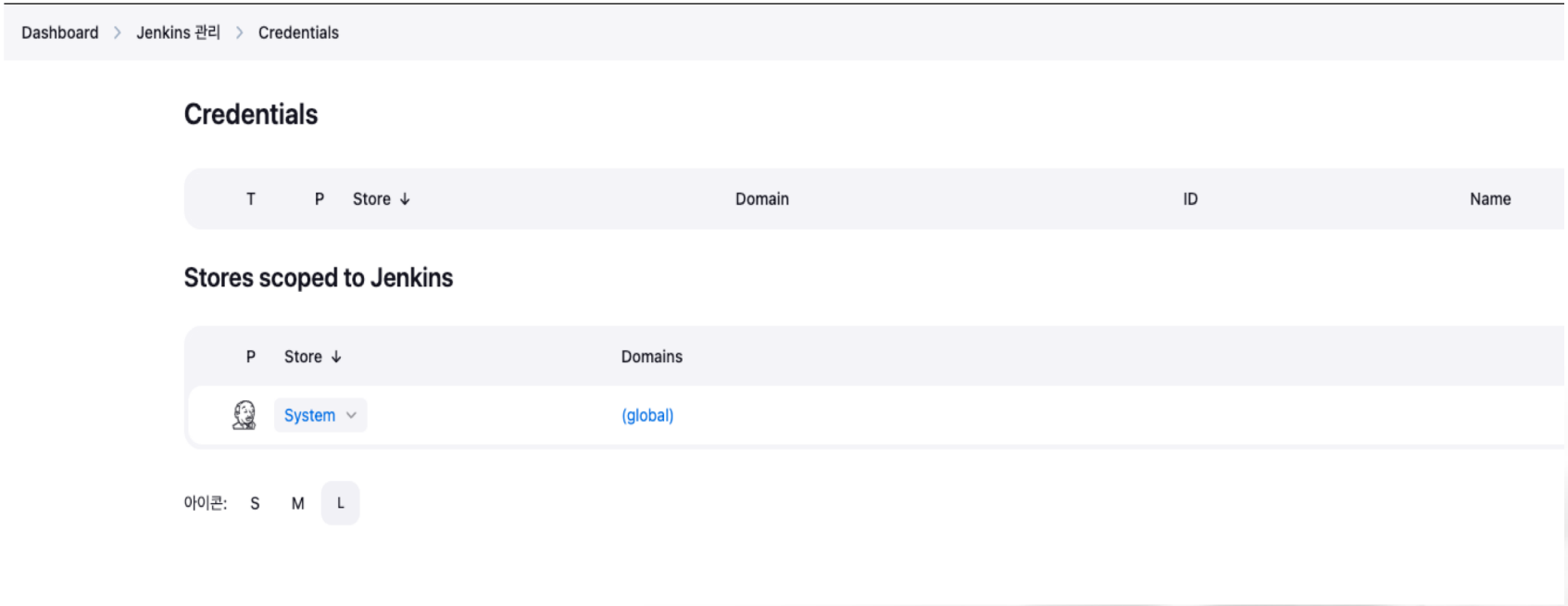
Docker Image 배포

- 도커 허브에 이미지 배포
  - ✓ 도커 허브 접속을 위한 Credential 생성
    - 도커 허브에서 토큰 생성
    - 도커 허브에 레포지토리 생성

Docker Image 배포

❖ 도커 허브에 이미지 배포

✓ Jenkins 서버에서 DockerHub 접속을 위한 Credential 생성: System 클릭



Docker Image 배포

- 도커 허브에 이미지 배포
  - ✓ Item의 구성 변경: execute shell 수정

```
chmod +x ./gradlew
./gradlew compileJava
./gradlew clean build

echo "Logging into Docker Hub"
DOCKER_USERNAME="ggnagpae1"
DOCKER_PASSWORD="dckr_pat_GN3R10FhAJ28yAAKwBlRwdkObh8"
# 비대화형 Docker 로그인
echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --password-stdin
echo "Logging into Docker Hub"

docker build -t ggnagpae1/calculator .
docker push ggnagpae1/calculator
```

Docker Image 배포

- ❖ 도커 허브에 이미지 배포
  - ✓ Jenkins를 Docker에 설치한 경우 docker not found 에러 제거
    - jenkins container 접속

      docker exec -it -u root jenkins /bin/bash

Docker Image 배포

- ❖ 도커 허브에 이미지 배포
  - ✓ Jenkins를 Docker에 설치한 경우 docker not found 에러 제거
    - ● 도커 설치

```
# linux 버전 확인
cat /etc/issue

# Docker 설치
## - Setup Repo
apt-get update

apt-get install ₩
    ca-certificates ₩
    curl ₩
    gnupg ₩
    lsb-release

mkdir -p /etc/apt/keyrings
```

Docker Image 배포

- ❖ 도커 허브에 이미지 배포
  - ✓ Jenkins를 Docker에 설치한 경우 docker not found 에러 제거
    - ● 도커 설치

```
#linux 버전에 맞게 설치
curl -fsSL https://download.docker.com/linux/debian/gpg | gpg --dearmor -o
    /etc/apt/keyrings/docker.gpg

echo ₩
  "deb [arch=$(dpkg --print-architecture) signed-
    by=/etc/apt/keyrings/docker.gpg] https://download.docker.com/linux/debian
    ₩
  $(lsb_release -cs) stable" | tee /etc/apt/sources.list.d/docker.list > /dev/null

## - Install Docker Engine
apt-get update

apt-get install docker-ce docker-ce-cli containerd.io docker-compose-plugin
```

Docker Image 배포

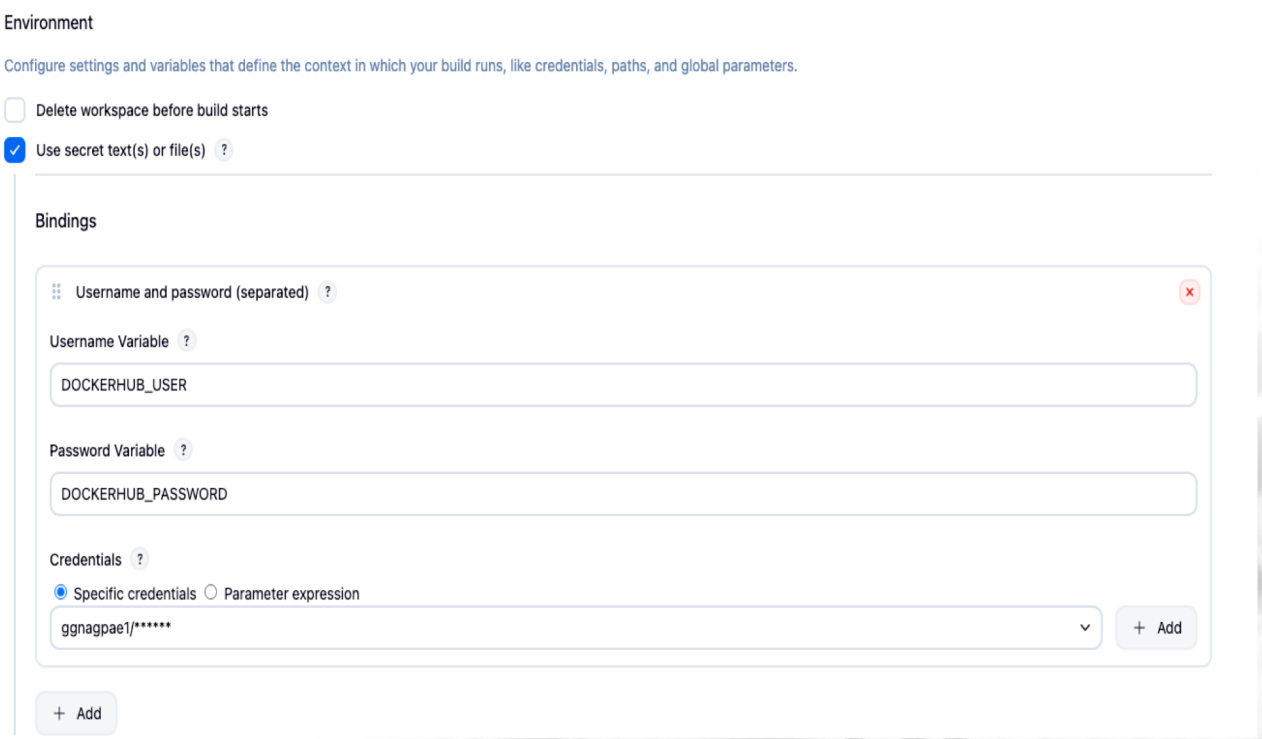
- ❖ 도커 허브에 이미지 배포
  - ✓ Jenkins를 Docker에 설치한 경우 docker.sock 퍼미션 에러
    - jenkins container 접속

      docker exec -it -u root jenkins /bin/bash
    - 권한 변경

      chmod 666 /var/run/docker.sock

Docker Image 배포

- ❖ 도커 허브에 이미지 배포
  - ✓ Credential 이용
    - 젠킨스 프로젝트 설정 페이지로 이동(구성)
    - 중간에 있는 Environment 섹션으로 이동
    - Use secret text(s) or file(s) 항목을 체크



Docker Image 배포

- ❖ 도커 허브에 이미지 배포
  - ✓ Credential 이용
    - Execute Shell 수정

```
chmod +x ./gradlew
./gradlew compileJava
./gradlew clean build

echo "Logging into Docker Hub"
DOCKER_USERNAME=$DOCKERHUB_USER
DOCKER_PASSWORD=$DOCKERHUB_PASSWORD
# 비대화형 Docker 로그인
echo "$DOCKER_PASSWORD" | docker login -u "$DOCKER_USERNAME" --
    password-stdin
echo "Logging into Docker Hub"

docker build -t ggnagpae1/calculator .
docker push ggnagpae1/calculator
```