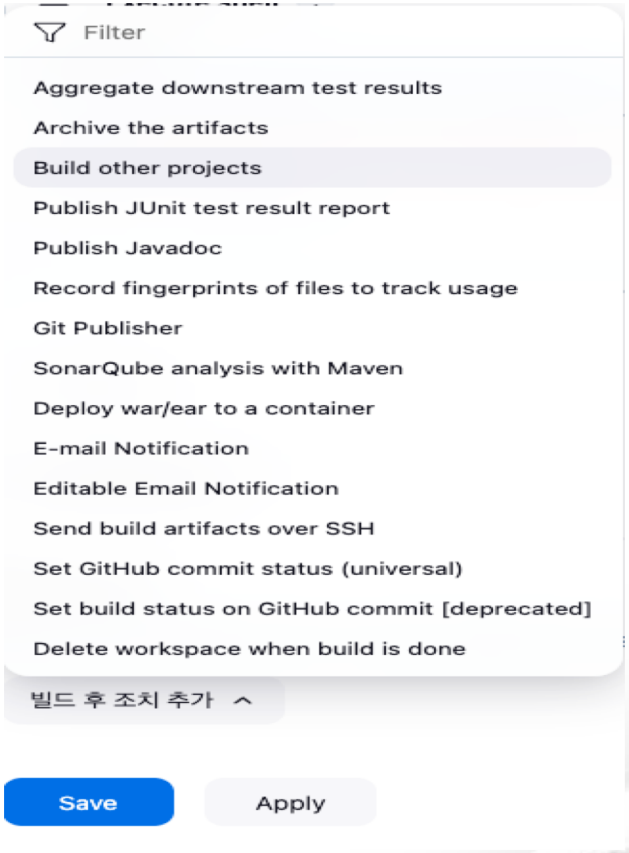
Jenkins Pipeline

Pipeline

- Pipeline 개요
  - ✓ Jenkins의 파이프라인이란 연속적인 이벤트 혹은 Job의 그룹
  - ✓ Jenkins Job들을 순차적 혹은 병렬적으로 실행시키거나 특별하게 작성한 스크립트로 이벤트들을 연속적으로 실행시키는 등의 일을 지원하는 기능
  - ✓ Jenkins Pipeline은 Jenkins를 사용하여 연속적인 전달 파이프라인의 통합 및 구현을 지원하는 플러그인의 조합
  - ✓ 파이프라인은 파이프라인 DSL(Domain-Specific Language)을 통해 간단하거나 복잡한 전달 파이프 라인을 코드로 생성 할 수 있는 확장 가능한 자동화 서버를 갖추고 있음

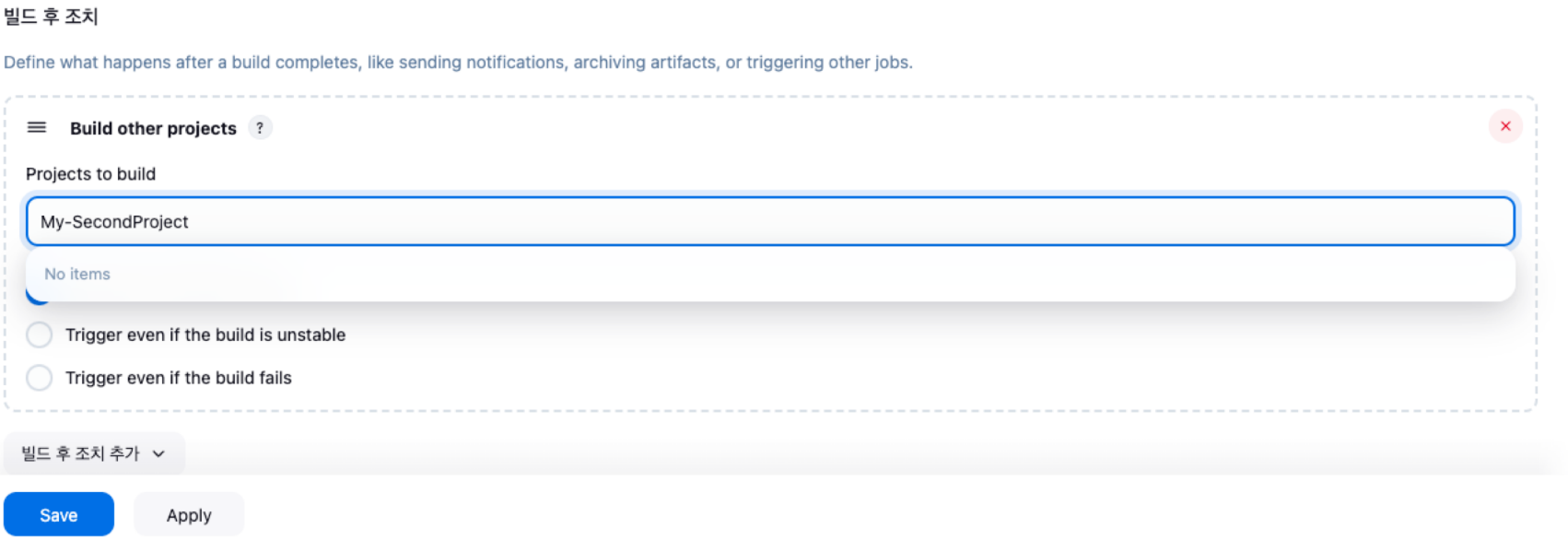
Pipeline

- 빌드 후 조치를 이용한 다른 프로젝트 빌드
  - ✓ 첫번째 프로젝트에서 [빌드 후 조치 추가]를 설정



Pipeline

- 빌드 후 조치를 이용한 다른 프로젝트 빌드
  - ✓ 첫번째 프로젝트에서 [빌드 후 조치 추가]를 설정

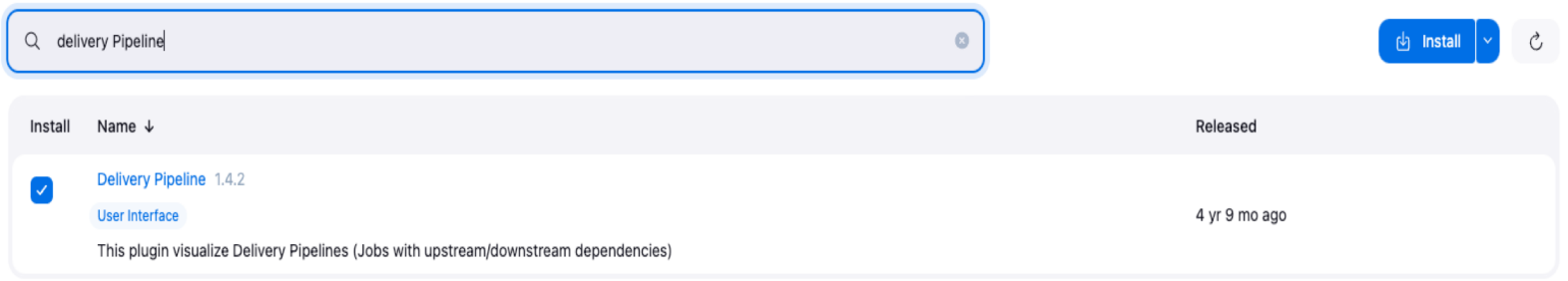


Pipeline

- 빌드 후 조치를 이용한 다른 프로젝트 빌드
  - ✓ 첫번째 프로젝트를 빌드하고 두번째 프로젝트 확인

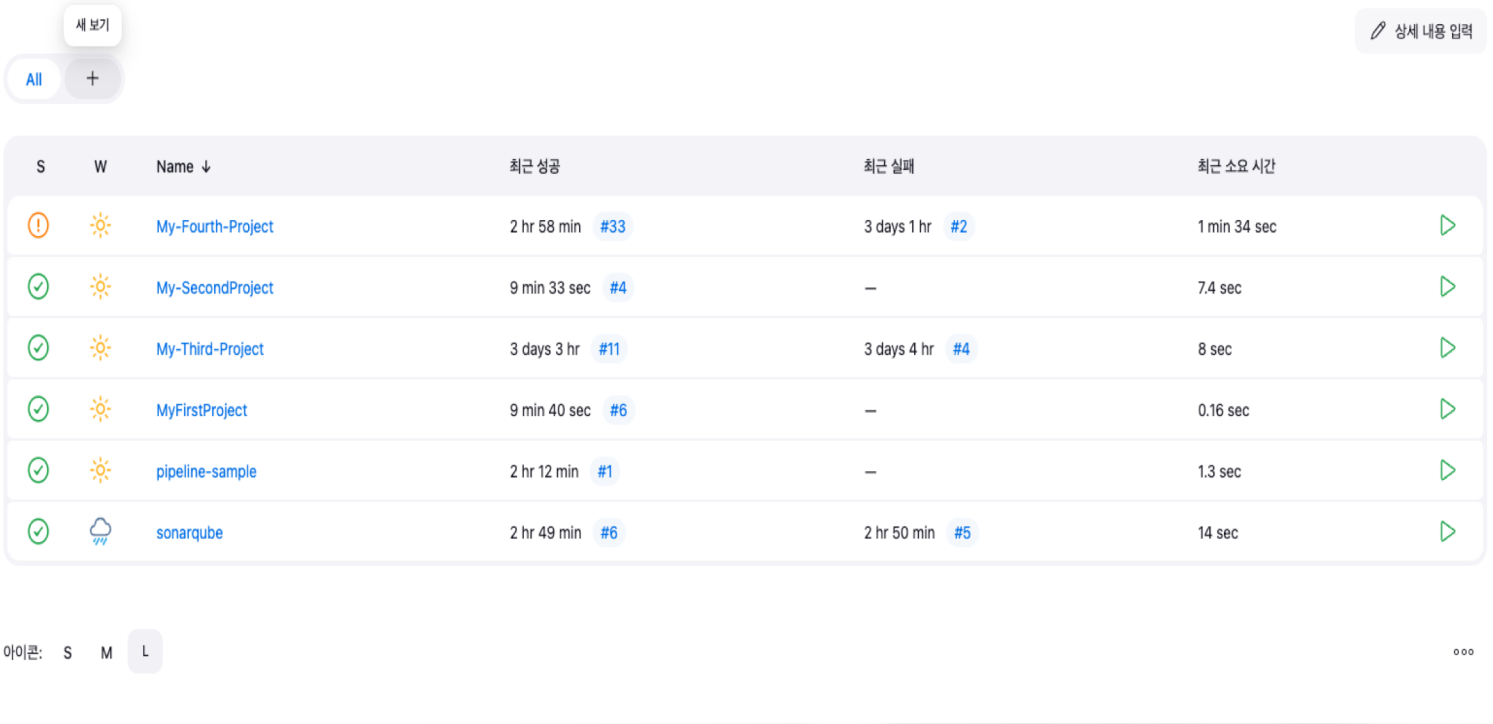
Pipeline

- Pipeline 시각화
  - ✓ 시각화 작업을 위한 플러그인 설치: Delivery Pipeline



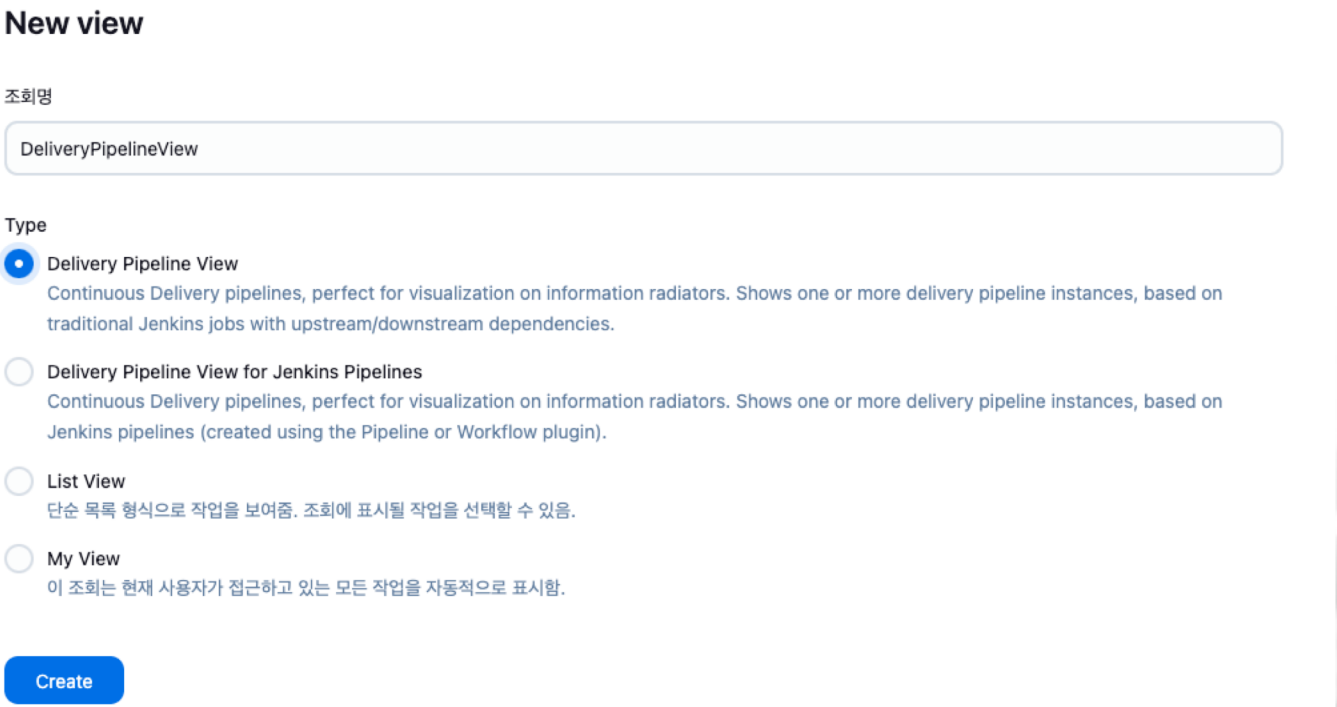
Pipeline

- Pipeline 시각화
  - ✓ Delivery Pipeline View 생성



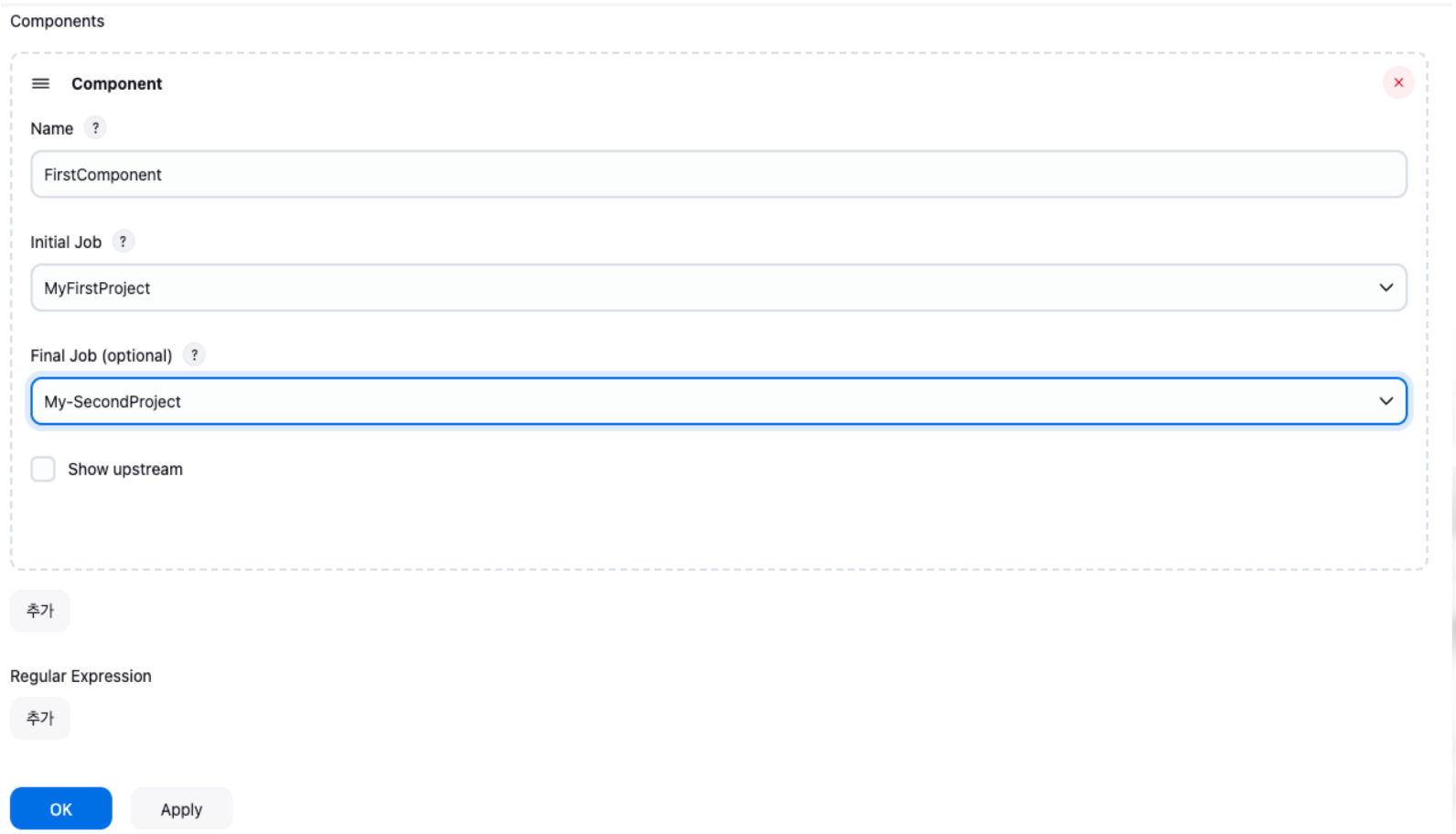
Pipeline

- ❖ Pipeline 시각화
  - ✓ Delivery Pipeline View 생성



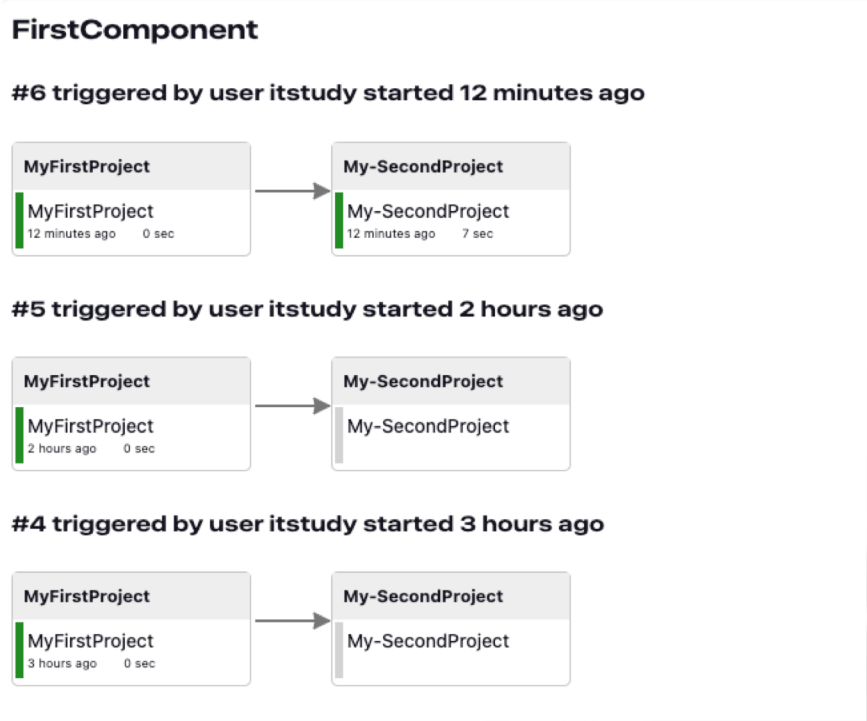
Pipeline

- ❖ Pipeline 시각화
  - ✓ Delivery Pipeline View 생성



Pipeline

- Pipeline 시각화
  - ✓ Delivery Pipeline View 생성

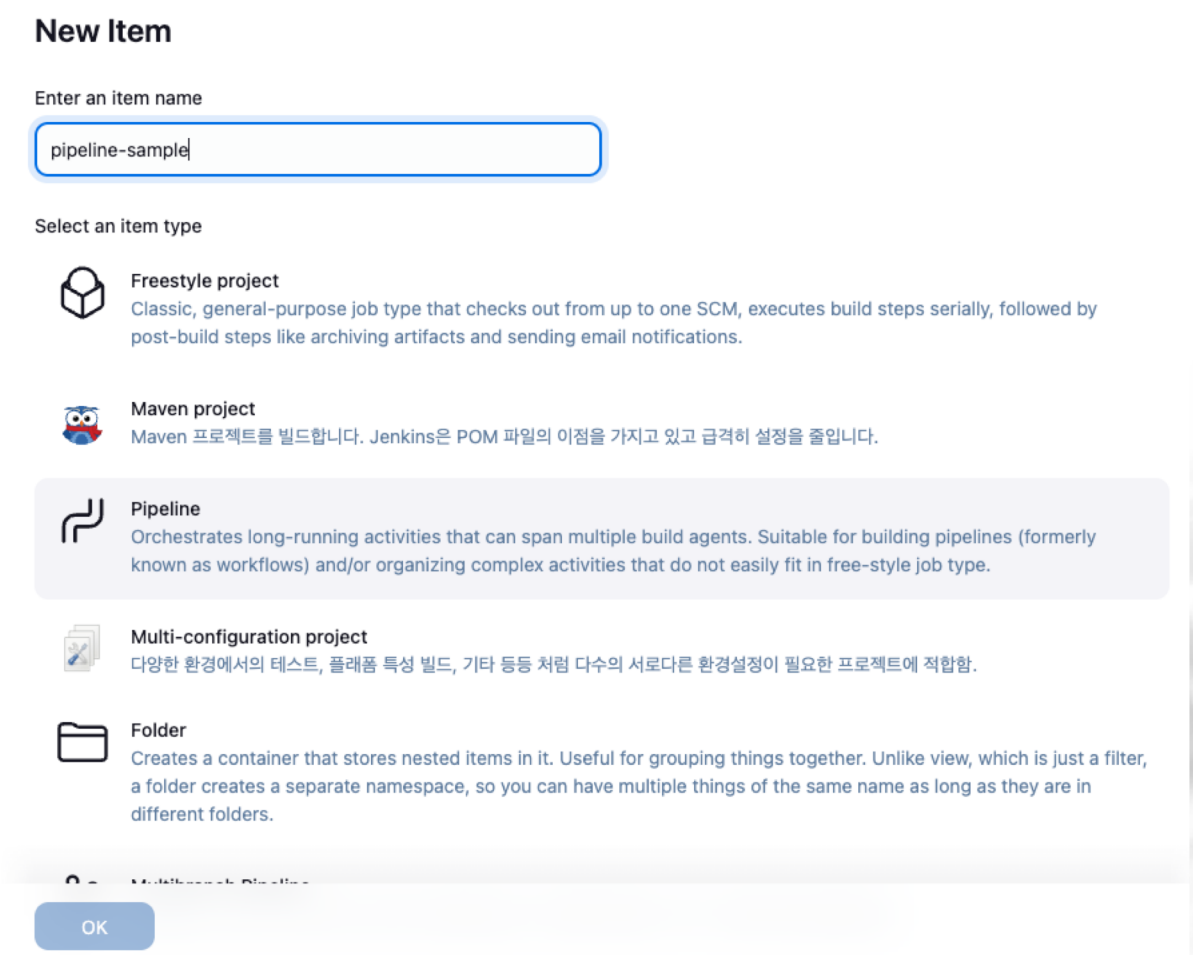


Pipeline

- Pipeline 프로젝트
  - ✓ 프로젝트 작성 방법
    - Declarative
    - Scripted (Groovy + DSL)
  - ✓ 차이점
    - 특정 Stage 실행 가능 여부(실패 시 다음 작업 수행: Scripted)
    - Scripted 방식은 제어문 사용 가능
    - Scripted 방식은 Pipeline으로 시작하고 Declarative 방식은 node로 시작

Pipeline

- Pipeline 프로젝트 생성 및 빌드
  - ✓ 파이프라인 프로젝트 생성



Pipeline

- Pipeline 프로젝트 생성 및 빌드
  - ✓ Declarative 방식
    - 스크립트 코드 작성

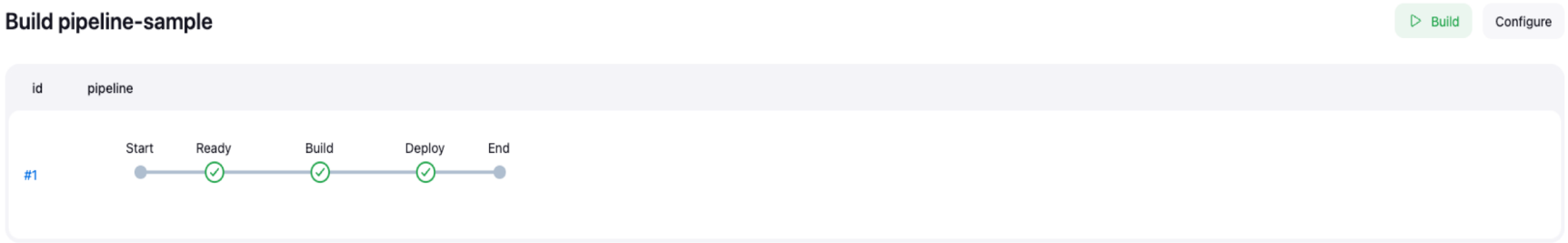
```
node {
    stage('Ready') {
        sh "echo 'Ready'"
    }

    stage('Build') {
        sh "echo 'Build Jar'"
    }

    stage('Deploy') {
        sh "echo 'Deploy AWS'"
    }
}
```

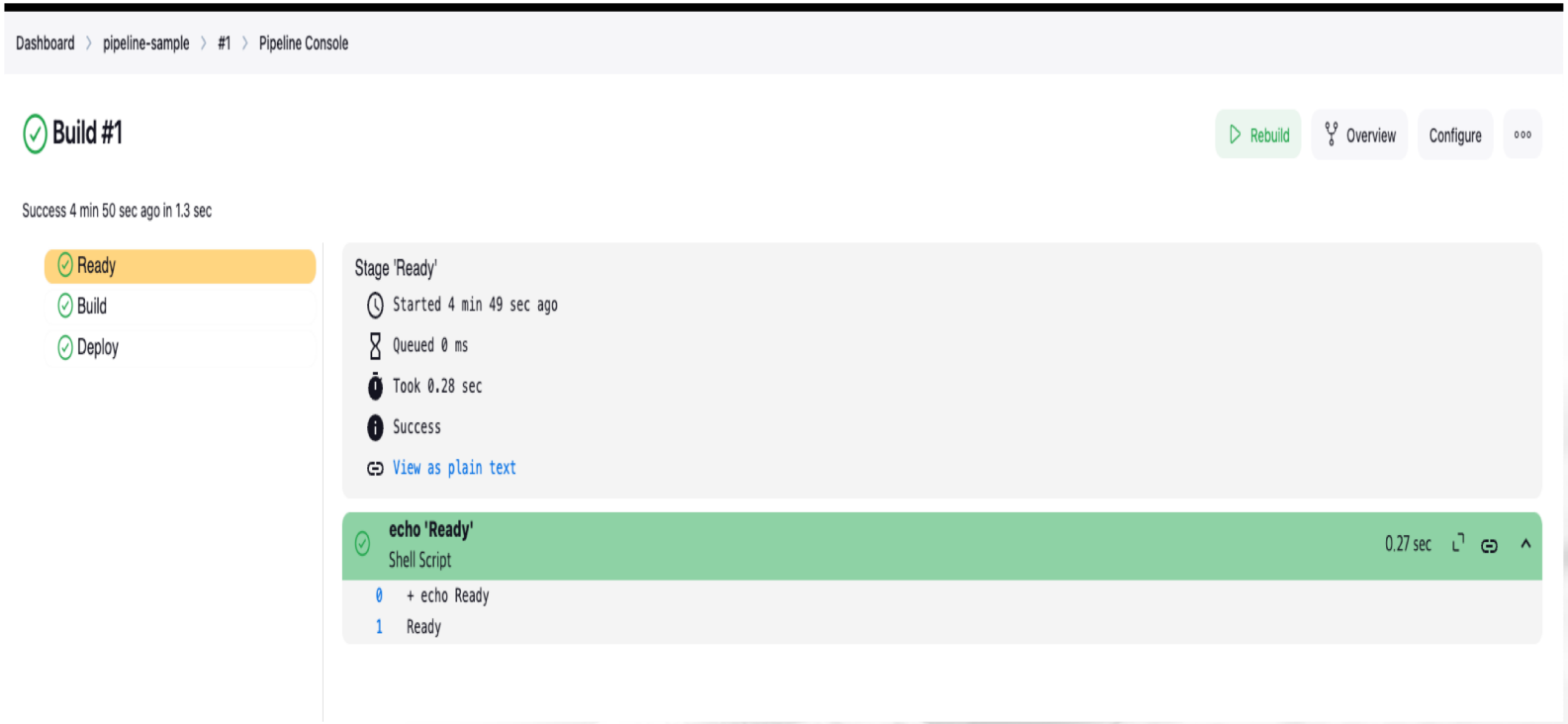
Pipeline

- ❖ Pipeline 프로젝트 생성 및 빌드
  - ✓ Declarative 방식
    - 빌드
    - 결과 확인



Pipeline

- ❖ Pipeline 프로젝트 생성 및 빌드
  - ✓ Declarative 방식
    - ● 결과 확인



Pipeline

- Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - 스크립트 코드 작성

```
pipeline {
    agent any
    stages{
        stage('Compile') {
            steps{
                echo 'Compile Successfully'
            }
        }

        stage('Junit') {
            steps{
                echo 'JUnit Passed Successfully'
            }
        }
```

Pipeline

- Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - 스크립트 코드 작성

```
        stage('Code Analysis') {
            steps{
                echo 'Code Analysis Completed Successfully'
            }
        }

        stage('Deploy') {
            steps{
                echo 'Deployed Successfully'
            }
        }
    }
}
```

Pipeline

- ❖ Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - ● 다시 빌드

Pipeline

- ❖ Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - ● 결과 출력

```
Started by user itstudy
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipeline-sample
[Pipeline] {
[Pipeline] stage
[Pipeline] { (Compile)
[Pipeline] echo
Compile Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Junit)
[Pipeline] echo
JUnit Passed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Code Analysis)
[Pipeline] echo
Code Analysis Completed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deployed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Pipeline

- Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - 스크립트 코드 추가 – stages 다음에 추가

```
post{
        always{
            echo 'This will always run'
        }
        success{
            echo 'This will run when finished successfully'
        }
        failure{
            echo 'This will run if failed'
        }
        unstable{
            echo 'This will run when the run was marked as unstable'
        }
        changed{
            echo 'This will run when the state of the pipeline has changed'
        }
    }
```

Pipeline

- ❖ Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - ● 다시 빌드

Pipeline

- ❖ Pipeline 프로젝트 생성 및 빌드
  - ✓ Scripted 방식
    - ● 결과 출력

```
Deployed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Declarative: Post Actions)
[Pipeline] echo
This will always run
[Pipeline] echo
This will run when finished successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

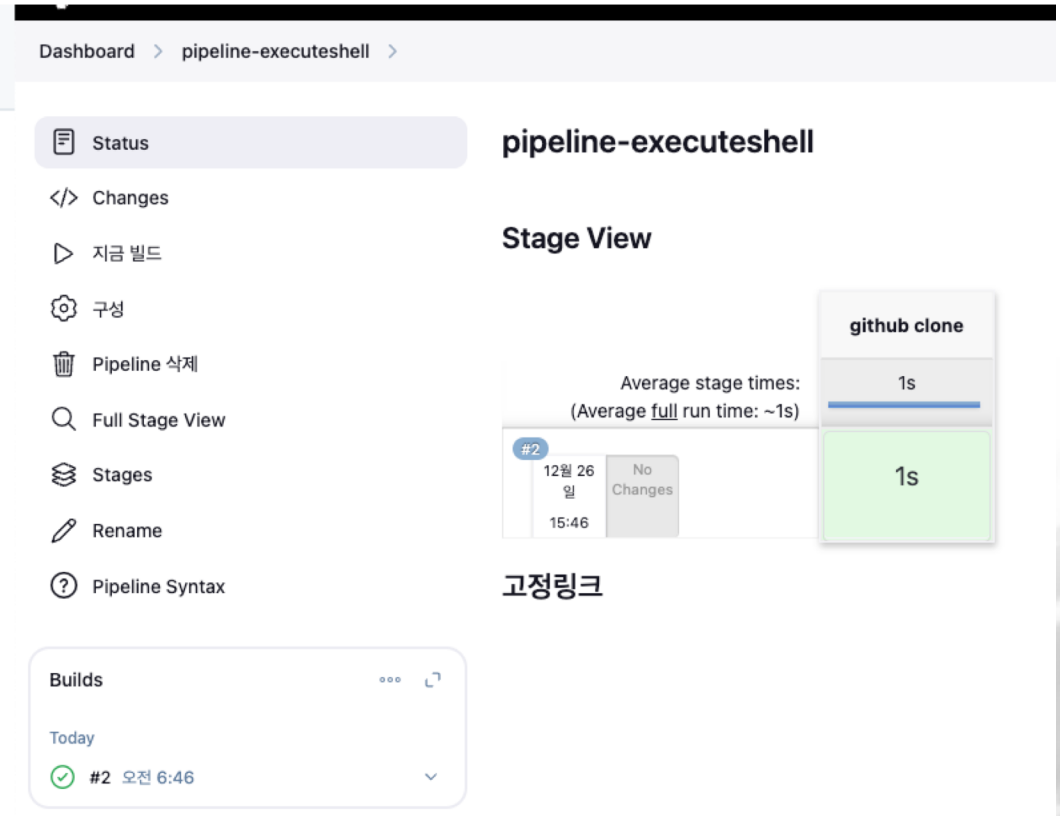
Pipeline

- Shell 명령어 수행
  - ✓ Git Clone

```
pipeline {
    agent any
    stages {
        stage('github clone') {
            steps {
                git branch: 'main', url: 'https://github.com/itggangpae/executeshell.git'
            }
        }
    }
}
```

Pipeline

- ❖ Shell 명령어 수행
  - ✓ Git Clone



Pipeline

- ❖ Shell 명령어 수행
  - ✓ Git Clone

```
Started by user itstudy
[Pipeline] Start of Pipeline
[Pipeline] node
Running on Jenkins in /var/jenkins_home/workspace/pipeline-executeshell
[Pipeline] {
[Pipeline] stage
[Pipeline] { (github clone)
[Pipeline] git
The recommended git tool is: NONE
No credentials specified
 > git rev-parse --resolve-git-dir /var/jenkins_home/workspace/pipeline-executeshell/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/itggangpae/executeshell.git # timeout=10
Fetching upstream changes from https://github.com/itggangpae/executeshell.git
 > git --version # timeout=10
 > git --version # 'git version 2.39.5'
 > git fetch --tags --force --progress -- https://github.com/itggangpae/executeshell.git +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/main^{commit} # timeout=10
Checking out Revision 8fac5231d221d8d6d13f309618457011fa59c157 (refs/remotes/origin/main)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 8fac5231d221d8d6d13f309618457011fa59c157 # timeout=10
 > git branch -a -v --no-abbrev # timeout=10
 > git checkout -b main 8fac5231d221d8d6d13f309618457011fa59c157 # timeout=10
Commit message: "commit"
First time build. Skipping changelog.
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Pipeline

- Shell 명령어 수행
  - ✓ Git에 있는 스크립트 파일을 가지고 수행
    - ● 원격 레포지토리에 Jenkinsfile을 생성하고 작성

```
pipeline {
    agent any
    stages{
        stage('Compile') {
            steps{
                echo 'Compile Successfully'
            }
        }

        stage('Junit') {
            steps{
                echo 'JUnit Passed Successfully'
            }
        }

        stage('Code Analysis') {
            steps{
                echo 'Code Analysis Completed Successfully'
            }
        }
```

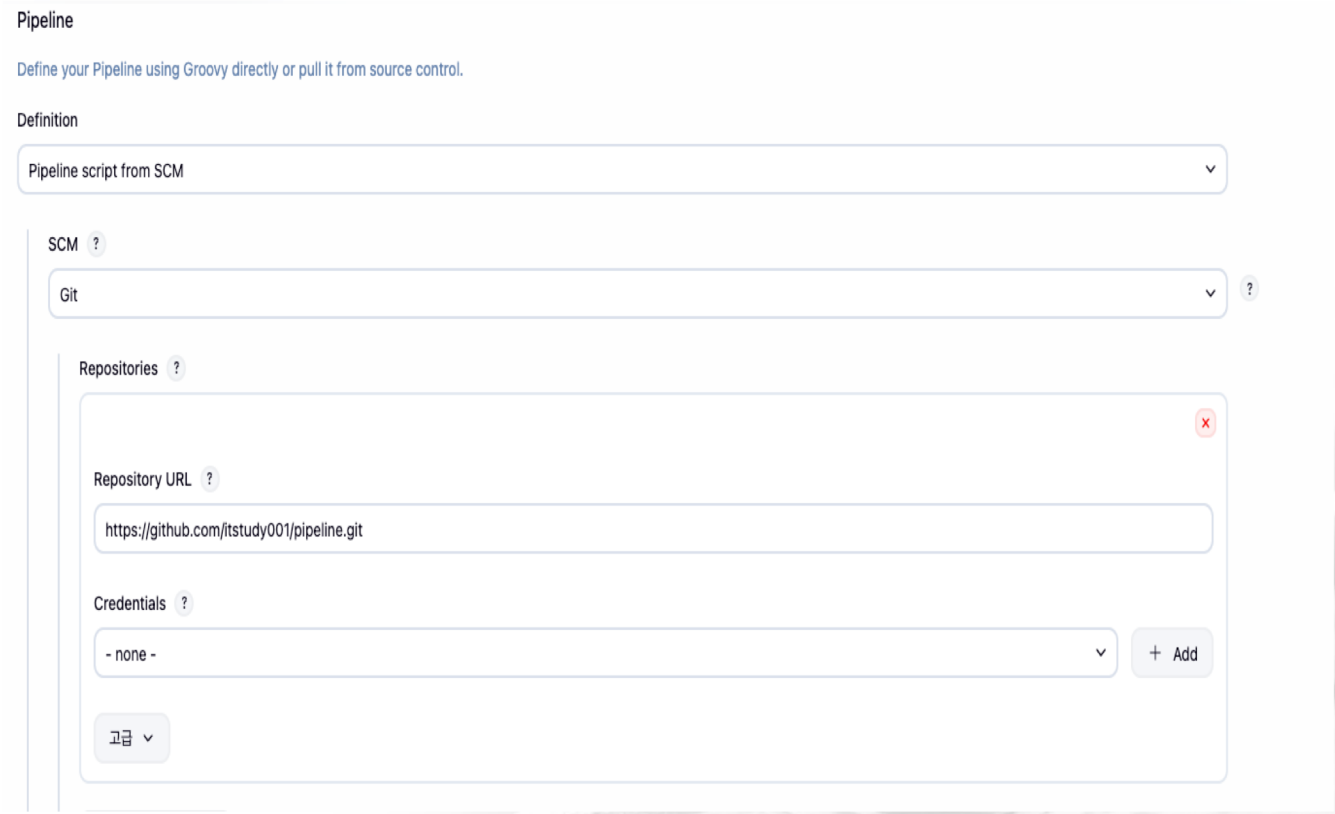
Pipeline

- Shell 명령어 수행
  - ✓ Git에 있는 스크립트 파일을 가지고 수행
    - ● 원격 레포지토리에 Jenkinsfile을 생성하고 작성

```
        stage('Deploy') {
            steps{
                echo 'Deployed Successfully'
            }
        }
    }
}
```

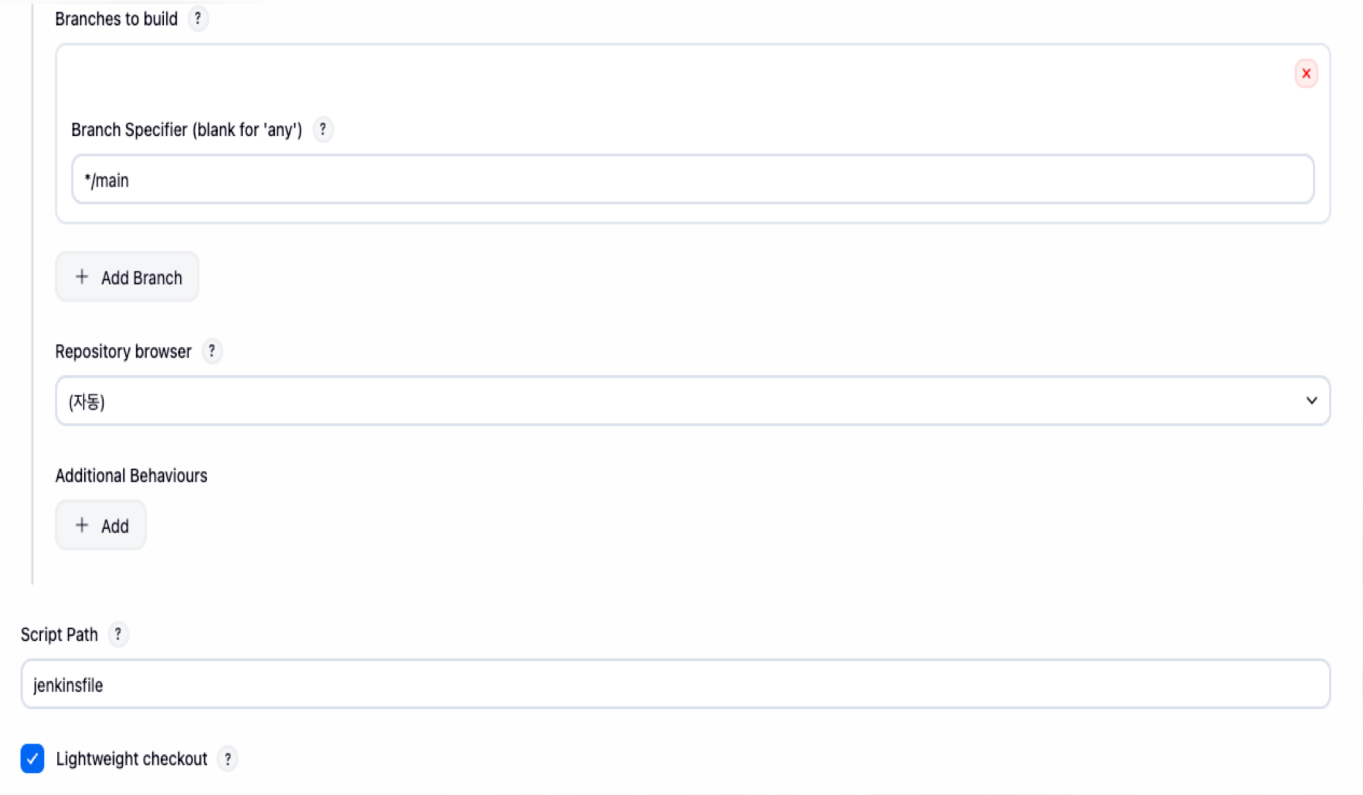
Pipeline

- ❖ Shell 명령어 수행
  - ✓ Git에 있는 스크립트 파일을 가지고 수행
    - ● 프로젝트의 구성 변경 – Pipeline 항목



Pipeline

- ❖ Shell 명령어 수행
  - ✓ Git에 있는 스크립트 파일을 가지고 수행
    - ● 프로젝트의 구성 변경 – Pipeline 항목

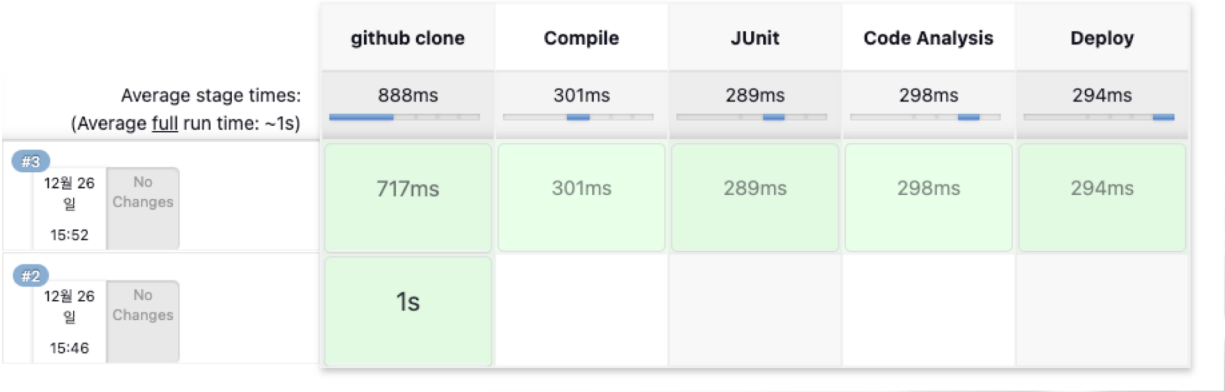


Pipeline

- ❖ Shell 명령어 수행
  - ✓ Git에 있는 스크립트 파일을 가지고 수행
    - ● 빌드

pipeline-executeshell

Stage View



Pipeline

- Shell 명령어 수행
  - ✓ Git에 있는 스크립트 파일을 가지고 수행
    - 빌드

```
Compile Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Junit)
[Pipeline] echo
JUnit Passed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Code Analysis)
[Pipeline] echo
Code Analysis Completed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] stage
[Pipeline] { (Deploy)
[Pipeline] echo
Deployed Successfully
[Pipeline] }
[Pipeline] // stage
[Pipeline] }
[Pipeline] // withEnv
[Pipeline] }
[Pipeline] // node
[Pipeline] End of Pipeline
Finished: SUCCESS
```

Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Git Clone
    - ● script 작성: 이전에 만들어 둔 프로젝트 가져오기

```
pipeline {
    agent any
    stages {
        stage('github clone') {
            steps {
                git branch: 'main', url: 'https://github.com/itggangpae/gradle-cicd-
project.git'
            }
        }
    }
}
```

Pipeline

- 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Build 스테이지 추가
    - script 추가

```
stage('Build') {
        steps {
            sh "./gradlew clean build --stacktrace"
        }
    }
```

Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Image Build
    - ● 프로젝트에 Dockerfile이 없으면 생성

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

Pipeline

- 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Image Build
    - 이미지 빌드 stage 추가

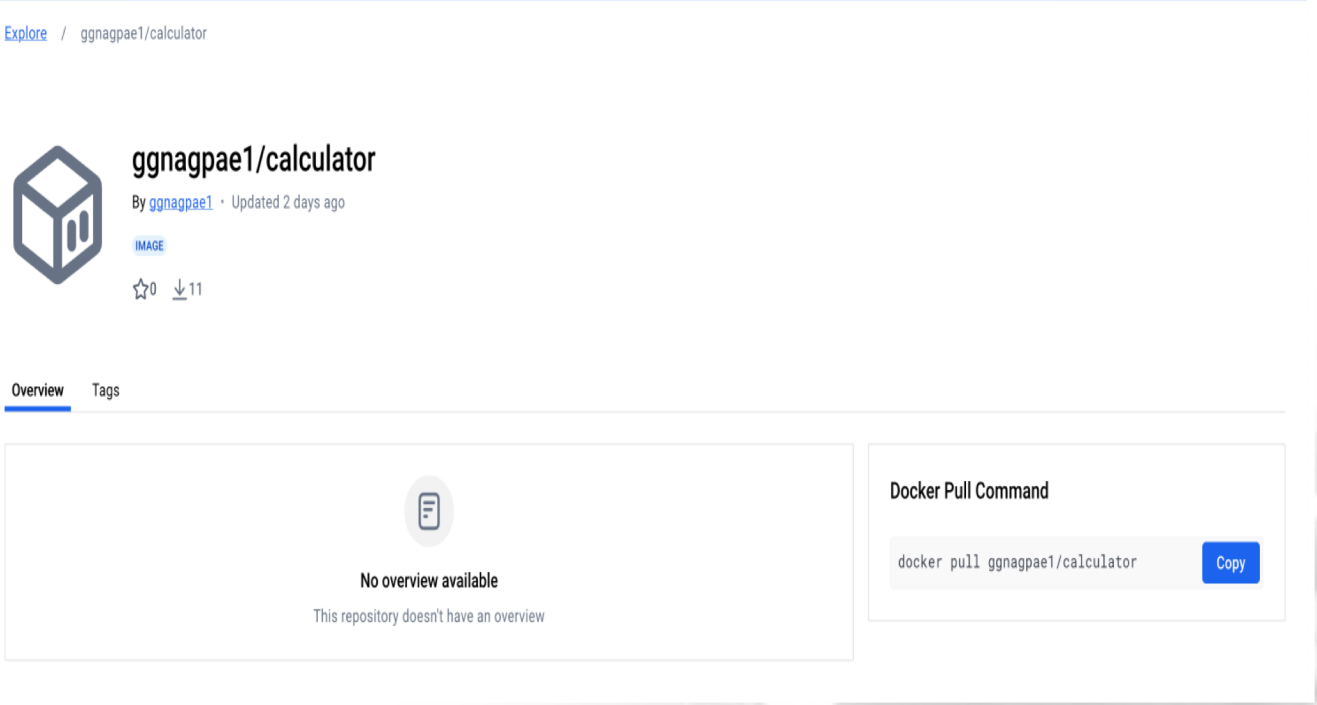
```
stage('Image Build') {
        steps {
            sh "docker build -t ggnagpae1/calculator ."
        }
    }
```

Pipeline

❖ 원격 서버에 도커 이미지로 애플리케이션 배포

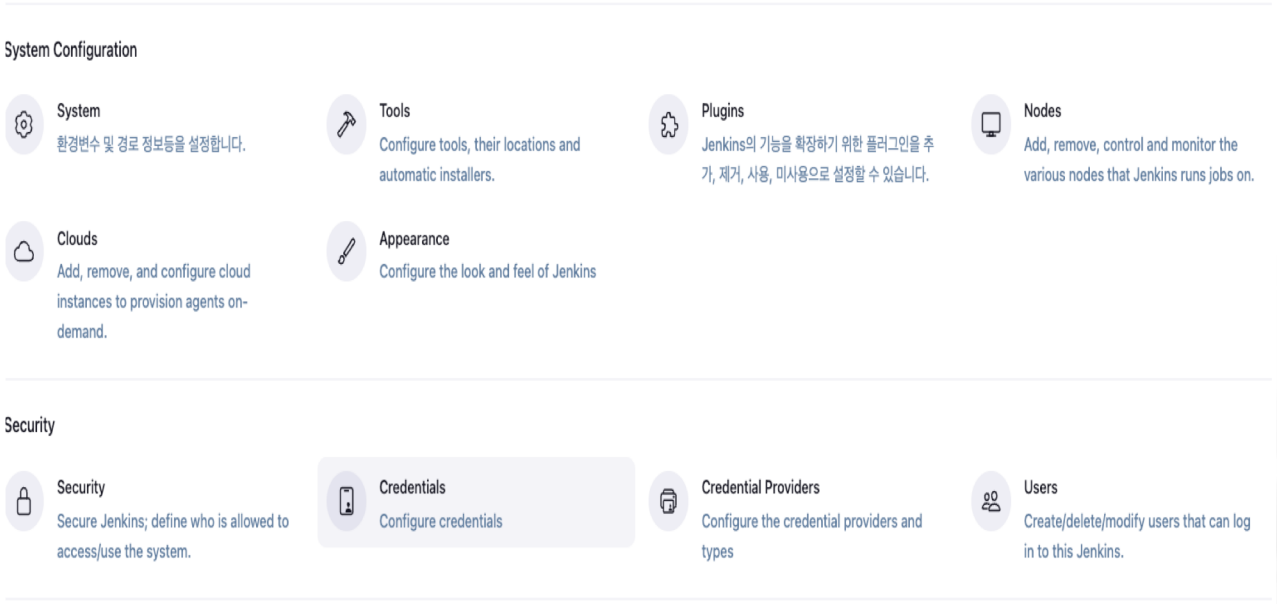
✓ Docker Hub Push

- Docker Hub에 접속해서 Access Token 과 Repository 생성



Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Docker Hub Push
    - ● Docker Hub에 접속하기 위한 Credential을 Jenkins Server에 생성

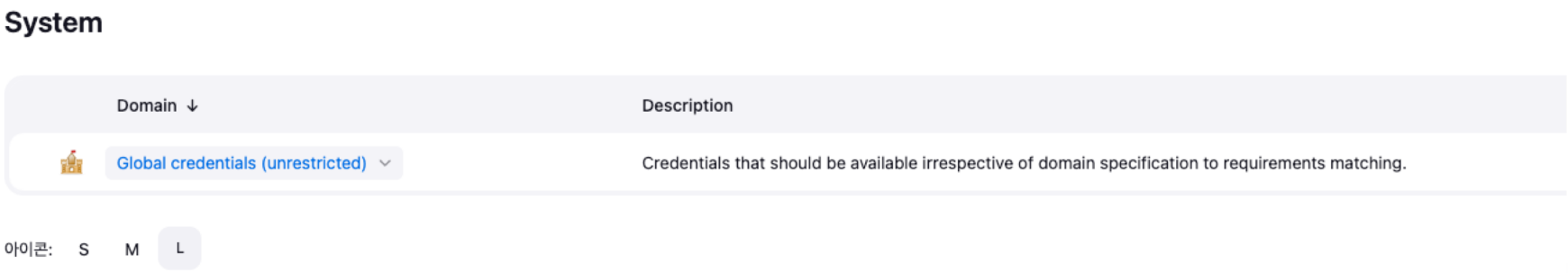


Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Docker Hub Push
    - ● Docker Hub에 접속하기 위한 Credential을 Jenkins Server에 생성

Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Docker Hub Push
    - ● Docker Hub에 접속하기 위한 Credential을 Jenkins Server에 생성

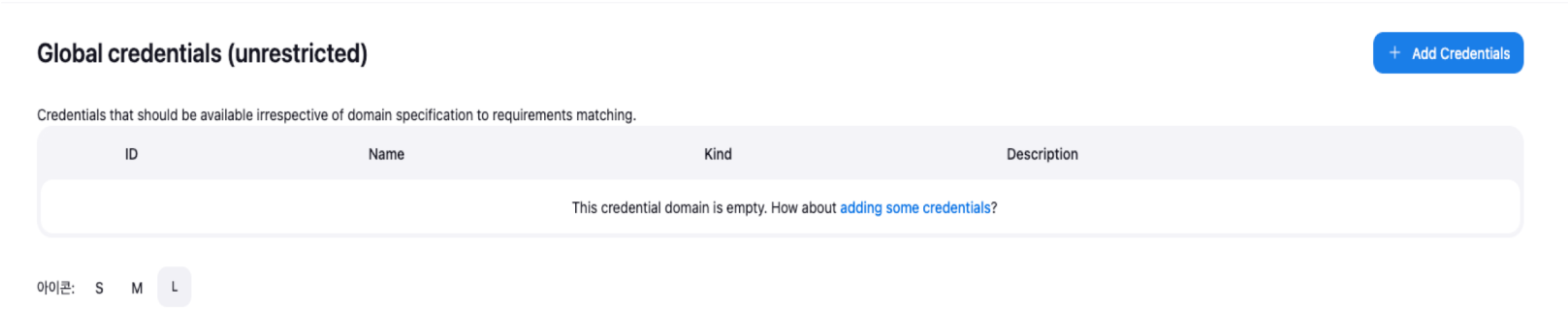


Pipeline

❖ 원격 서버에 도커 이미지로 애플리케이션 배포

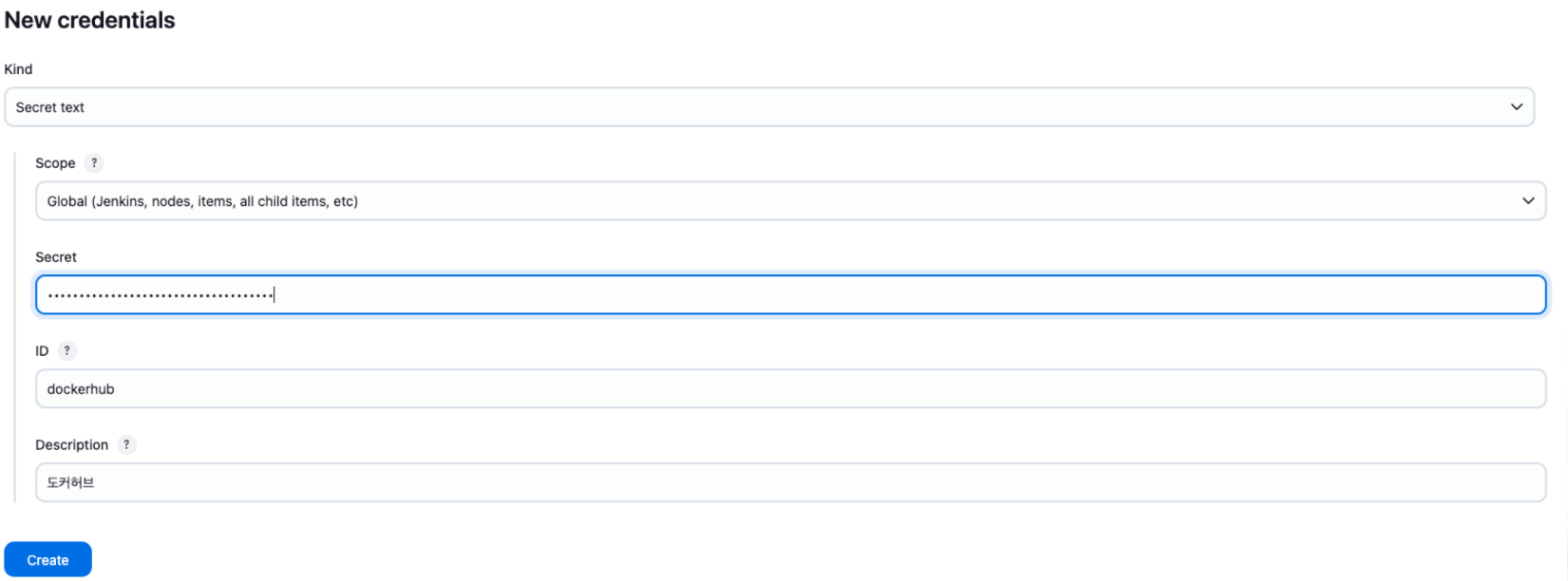
✓ Docker Hub Push

- Docker Hub에 접속하기 위한 Credential을 Jenkins Server에 생성



Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Docker Hub Push
    - ● Docker Hub에 접속하기 위한 Credential을 Jenkins Server에 생성



Pipeline

❖ 원격 서버에 도커 이미지로 애플리케이션 배포

✓ Docker Hub Push

- Script 추가

```
pipeline {
    agent any

    environment {
        DOCKER_CREDENTIALS_ID = 'docker-hub' // 저장한 자격 증명의 ID를 입력합
니다.
    }
    stages {
        stage('github clone') {
            steps {
                git branch: 'main', url: 'https://github.com/itggangpae/gradle-cicd-
project.git'
            }
        }
         stage('Build') {
            steps {
                sh "./gradlew clean build --stacktrace"
            }
        }
```

Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Docker Hub Push
    - ● 도커 허브 로그인 stage 추가

```
stage('Docker Login') {
            steps {
                script {
                    withCredentials([usernamePassword(credentialsId:
env.DOCKER_CREDENTIALS_ID, usernameVariable: 'DOCKER_USERNAME',
passwordVariable: 'DOCKER_PASSWORD')]) {
                        sh 'echo $DOCKER_PASSWORD | docker login -u
$DOCKER_USERNAME --password-stdin'
                    }
                }
            }
}
```

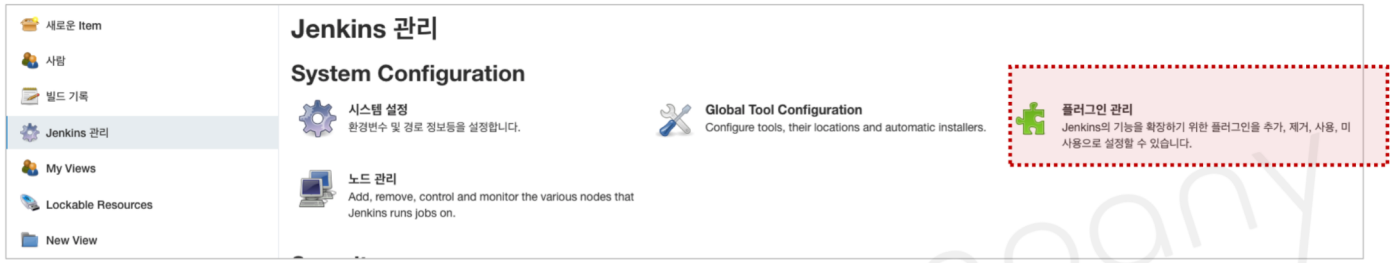
Pipeline

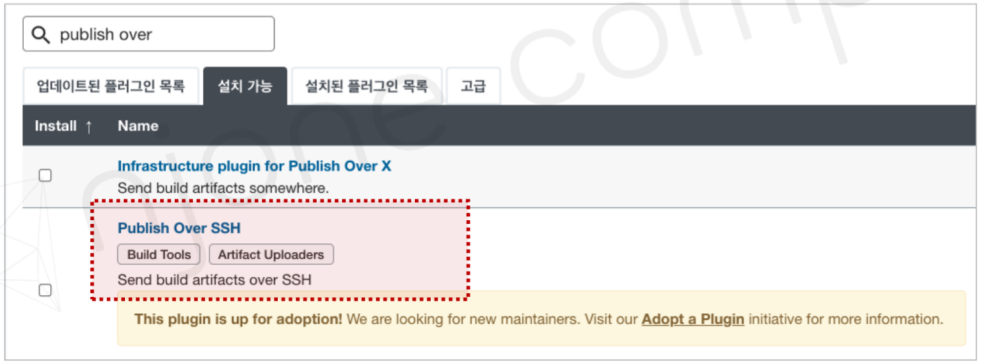
- 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ Docker Hub Push
    - 도커 허브 푸시 스테이지 추가

```
stage("Docker push"){
  steps{
    sh "docker push ggnagpae1/calculator"
  }
}
```

Pipeline

- 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ SSH Server에 배포
    - Jenkins에 publish Over SSH 와 SSH Agent 플러그인 설치



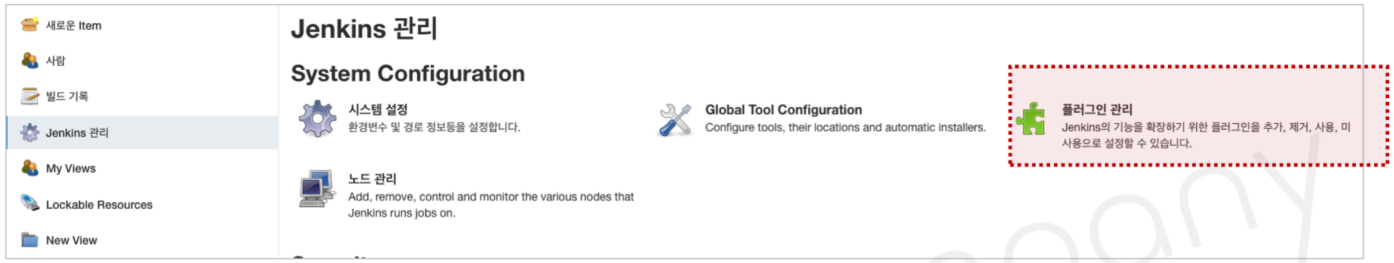


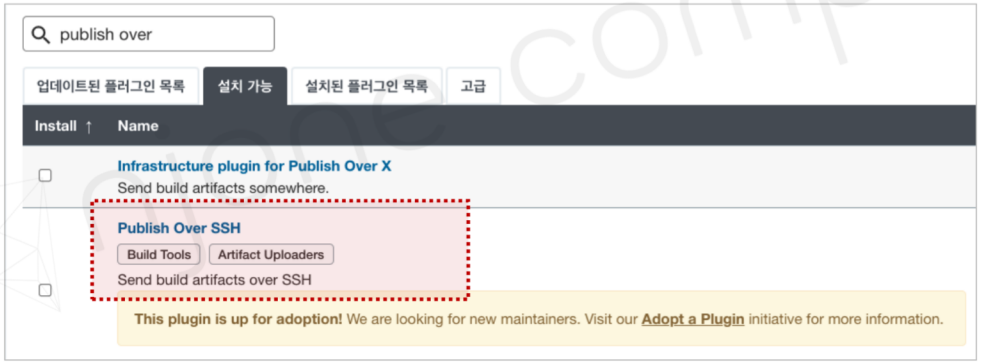
Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ SSH Server에 배포
    - ● SSH Server 생성
      - ◆ SSH 서비스가 구동 중 인 서버 준비
      - ◆ SSH 서버에 Docker 엔진 설치
      - ◆ SSH 접속 키 생성(EC2의 경우 pem 키값): ssh-keygen -t rsa -b 4096

Pipeline

- 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ SSH Server에 배포
    - Jenkins에 publish Over SSH 와 SSH Agent 플러그인 설치





Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ SSH Server에 배포
    - ● Jenkins에 SSH Credential 생성
      - ◆ Kind: SSH Username with private key를 선택
      - ◆ Scope: Global (기본값 유지)
      - ◆ ID: my-ssh-credentials (파이프라인 코드에서 호출할 이름이므로 정확히 입력)
      - ◆ Description: 이 자격 증명에 대한 간단한 설명(예: 웹 서버 SSH 접속용)
      - ◆ Username: 원격 서버에 접속할 계정 이름(예: ubuntu, ec2-user, root 등)
      - ◆ Private Key: Enter directly를 체크한 뒤 나타나는 Add 버튼을 클릭한 후 생성하거나 가지고 있는 Private Key 파일의 전체 내용(-----BEGIN RSA PRIVATE KEY----- 부터 끝까지)을 복사해서 붙여넣기
      - ◆ Passphrase: 키를 생성할 때 비밀번호를 설정했다면 입력하고 없으면 비워둠

Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ SSH Server에 배포
    - Git Repository에 sh 파일을 작성해서 업로드

```
IMAGE_NAME="ggnagpae1/calculator"
TAG="latest"

# 기존 이미지가 있는지 확인
if docker images | grep -q "$IMAGE_NAME"; then
    echo "이미지 $IMAGE_NAME:$TAG 삭제 중..."
    docker rmi -f "$IMAGE_NAME:$TAG"
fi

docker ps -q -f name=calculator | grep -q . && sudo docker stop calculator

# 새 이미지 다운로드
echo "이미지 $IMAGE_NAME:$TAG 다운로드 중..."
docker pull "$IMAGE_NAME:$TAG"

docker run -d -p 9090:9090 --rm --name calculator "$IMAGE_NAME:$TAG"
docker image prune -f
echo "작업 완료!"
```

Pipeline

- ❖ 원격 서버에 도커 이미지로 애플리케이션 배포
  - ✓ SSH Server에 배포
    - ● 배포를 위한 스테이지 추가

```
stage('test ssh') {
        steps {
            echo 'SSH'
            sshagent (credentials: ['my-ssh-credentials']) {
                sh "ssh -p 10004 -o StrictHostKeyChecking=no
adam@192.168.201.104 'git clone
https://github.com/itstudy001/jenkinsgradle.git'"
                sh "ssh -p 10004 -o StrictHostKeyChecking=no
adam@192.168.201.104 'chmod 777 jenkinsgradle/script.sh'"
                sh "ssh -p 10004 -o StrictHostKeyChecking=no
adam@192.168.201.104 'jenkinsgradle/script.sh'"
                sh "ssh -p 10004 -o
StrictHostKeyChecking=no adam@192.168.201.104 'rm -rf jenkinsgradle'"
            }
        }
    }
```