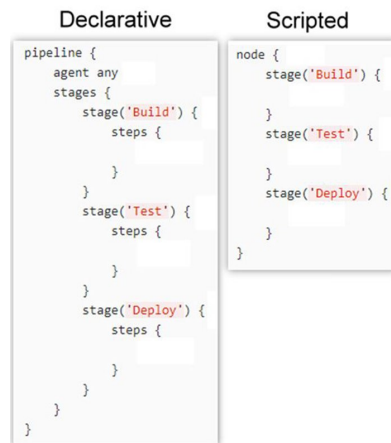


Declarative Pipeline

📅 생성일	@2021년 8월 15일
🏷 태그	작성완료

Jenkins 파이프라인에는 **Declarative Pipeline**과 **Scripted Pipeline** 2가지의 방식이 존재.



두 파이프라인 방식 모두 강력한 성능을 가지고 있지만, 좀 더 최신이자 가독성 좋은 문법을 가진 **Declarative Pipeline syntax**를 학습한다.

(둘다 공부해보고 업무 복잡도에 따라 어떤 방식을 선택할지 결정하는 것이 좋다.

간단한 작업의 경우에는 파이프라인보단 UI를 통해 정의하는 방식을 많이 사용하는 것 같다.)

Scripted 문법의 장점과 단점

Scripted 문법의 장점과 단점

장점

- 더 많은 절차적인 코드를 작성가능
- 프로그램 작성과 흡사
- 기존 파이프라인 문법이라 친숙하고 이전 버전과 호환가능
- 필요한 경우 커스텀한 작업 생성이 가능하기 때문에 유연성이 좋음
- 보다 복잡한 워크플로우 및 파이프라인 모델링 가능

단점

- 일반적으로 더 많은 프로그래밍이 필요
- Groovy 언어 및 환경으로 제한된 구문 검사
- 전통적인 젠킨스 모델과는 맞지 않음
- 같은 작업이라면 Declarative 문법보다 잠재적으로 더 복잡

Pipeline syntax

Sections

- Agent section
- Post section

- Stages section
- Steps section

으로 구성된다.

Agent Section

젠킨스는 많은 일들을 해야 하기 때문에 혼자 하기 버겁다.

여러 slave node를 두고 일을 시킬 수 있는데, 이처럼 어떤 젠킨스가 일을 하게 할 것인지 지정한다.

젠킨스 노드관리에서 새로 노드를 띄우거나 docker 이미지등을 통해서 처리할 수 있다.

※마스터 - 슬레이브를 구축하고자 할 경우 참고하면 좋은 Youtube 영상 링크※

https://www.youtube.com/watch?v=Tk7_KBxLUy0&list=PL3LtXuyXa_8_Z2OGFnJHGBjArORZDPo1V&index=6&t=763s

(해당 영상은 scripted pipeline 문법을 사용하기 때문에 설정만 참고한다)

Post section

스테이지가 끝난 이후, 결과에 따라 후속조치를 취할 수 있음.

Ex) success, failure, always, cleanup

Ex) 성공시에 성공 이메일, 실패하면 중단 혹은 건너뛰기 등등..

```
pipeline {
  agent any
  // Stages Section
  // 어떤 일들을 처리할 건지 일련의 stage를 정의한다.
  stages {
    stage('prepare') {
      // 한 스테이지 안에서의 단계로 일련의 스텝을 보여준다.
      steps {
        git url: 'https://github.com/sjabber/redteam_server.git',
            branch: 'master',
            credentialsId: 'sjabber'
      }
    }
  }

  post {
    success {
      echo 'success!!!'
    }
    failure {
      echo 'fail...'
    }
  }

  stage('Build') {
    steps {
      echo 'Building...'
    }
  }
}
```

Declaratives

Environment, stage, options, parameters, triggers, when 등의 Declarative가 있다.

Environment → 어떤 pipeline 이나 stage scope 의 환경 변수 설정

Parameter → 파이프라인 실행시 파라미터 받음

Triggers → 어떤 형태로 트리거 되는가

When → 언제 실행되는가


Triggers 설정 (PollSCM 체크 필요)

```
triggers {
  pollSCM('*/*3 * * * *')
  // 3분주기로 파이프라인을 구동한다.
}
```

When 문법 예제

```
pipeline {
  agent any
  stages {
    stage('test1 : Using when_01') {
      # when에서 설정한 조건이 True일 경우에만 steps이 실행된다.
      when { branch 'master' } # branch가 master인 경우 다음 steps이 실행됨
      steps {
        echo 'It is executed'
      }
    }
    stage('test1 : Using when_02') {
      when { not { branch 'master' } } # branch가 master가 아닐 경우 다음 steps이 실행됨
      steps {
        echo 'It is not executed'
      }
    }
    stage('test3 : Using Script') {
      steps {
        # script 의 {} 태그 안에서 Groovy를 사용할 수 있게 해준다
        script {
          if (env.BRANCH_NAME == 'master') { # branch가 master일 경우 if문 안의 코드가 실행
            for (i in 1 .. 5){ # 메시지를 5번 반복
              echo 'I only execute on the master branch'
            }
          } else { # branch가 master가 아닐 경우 if문 안의 코드가 실행
            echo 'I execute elsewhere'
          }
        }
      }
    }
  }
}
```

jenkins pipeline 문법표

<u>Aa</u> 문법	 의미
<u>pipeline</u>	젠킨스의 파이프라인 코드라는 것을 선언

Aa 문법	≡ 의미
<u>agent</u>	파이프라인을 빌드할 곳을 선택 any : 추가적인 설정이 없다면 기본 설정된 master agent에서 실행된다. none : 설정안함 ⇒ 따로 설정해주어야함
<u>stages</u>	파이프라인 작업의 단위로 stage를 위치시키는 블록 반드시 하나 이상의 stage를 포함해야한다.
<u>stage</u>	파이프라인의 작업(Check out, Test, Build, Deploy 등..)의 단위 반드시 1개 이상의 steps를 포함해야한다.
<u>steps</u>	stage에서 정의한 대로 작업을 정의하는 곳
<u>echo</u>	메시지를 출력한다. ex : echo 'hello world' ※ 변수의 값을 출력할 경우 echo "여기는 \${VALUE} 입니다" ⇒ ""을 사용해서 감싸야만한다 ⇒ " 안됨!
<u>sh</u>	shell script를 사용한다.
<u>writeFile</u>	파일을 생성
<u>\${VALUE}</u>	변수의 값을 사용할 경우
<u>environment</u>	환경변수 설정 블록
<u>parameters</u>	변수 설정 블록
<u>post</u>	작업이 모두 끝나고 나서의 처리

※ 자주 사용할 용어 정리

node / agent ⇒ 젠킨스를 사용하여 파이프라인들을 빌드 할때, 빌드되는 곳의 PC또는 도커

컨테이너 node의 경우 Scripted Pipeline의 문법과 표기가 겹치지만 여기서 다른 의미.











master : 젠킨스 서버가 설치된 PC 또는 도커 컨테이너를 의미한다.

slave : 젠킨스 서버(master)를 통해 리모트로 조작하는 PC 또는 도커 컨테이너

jenkins에 github, AWS EC2설정

Jenkins 관리 ⇒ Manage Credentials ⇒ Add Credentials

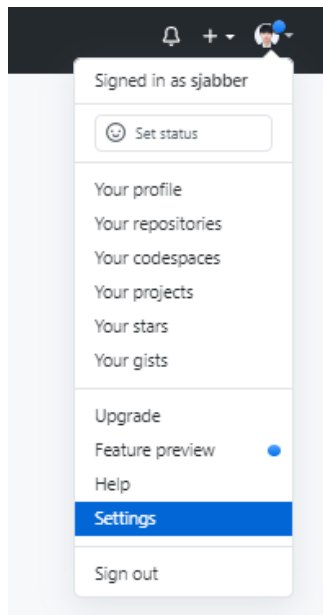
Credentials

T	P	Store ↓	Domain	ID	Name
		Jenkins	(global)	awsAccessKey	aws 액세스 키
		Jenkins	(global)	awsSecretAccessKey	aws 비밀 액세스 키
		Jenkins	(global)	sjabber	sjabber/***** (github 연동)
		Jenkins	(global)	dockerPasswd	도커허브 패스워드
		Jenkins	(global)	ec2-server	ubuntu (aws ec2)

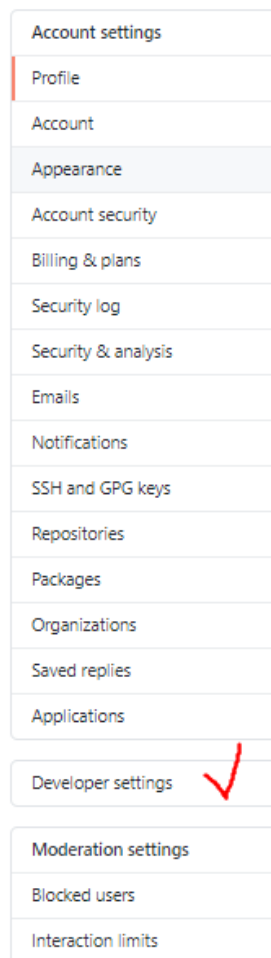
Credential들을 저장한 모습

github (Username with password)

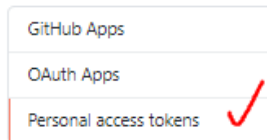
1. github에 로그인한다.
2. settings 클릭



3. Developer settings 클릭



4. Personal access tokens 클릭



5. Generate new token 클릭

⇒ 이후 repo, admin:repo_hook, admin:org_hook 체크 후 Generate token 클릭

Note

jenkins

What's this token for?

Expiration

This token has no expiration date. To set a new expiration date, you must [regenerate the token](#).

Select scopes

Scopes define the access for personal tokens. [Read more about OAuth scopes](#).

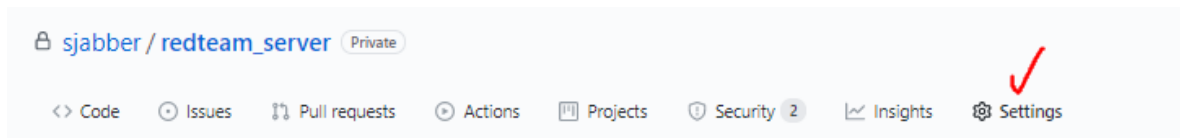
<input checked="" type="checkbox"/> repo	Full control of private repositories
<input checked="" type="checkbox"/> repo:status	Access commit status
<input checked="" type="checkbox"/> repo_deployment	Access deployment status
<input checked="" type="checkbox"/> public_repo	Access public repositories
<input checked="" type="checkbox"/> repo:invite	Access repository invitations
<input checked="" type="checkbox"/> security_events	Read and write security events
<input type="checkbox"/> workflow	Update GitHub Action workflows
<input type="checkbox"/> write:packages	Upload packages to GitHub Package Registry
<input type="checkbox"/> read:packages	Download packages from GitHub Package Registry
<input type="checkbox"/> delete:packages	Delete packages from GitHub Package Registry
<input type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input type="checkbox"/> read:org	Read org and team membership, read org projects
<input type="checkbox"/> admin:public_key	Full control of user public keys
<input type="checkbox"/> write:public_key	Write user public keys
<input type="checkbox"/> read:public_key	Read user public keys
<input checked="" type="checkbox"/> admin:repo_hook	Full control of repository hooks
<input checked="" type="checkbox"/> write:repo_hook	Write repository hooks
<input checked="" type="checkbox"/> read:repo_hook	Read repository hooks
<input checked="" type="checkbox"/> admin:org_hook	Full control of organization hooks

6. 발행된 토큰을 Add Credentials의 Password에 입력한다.

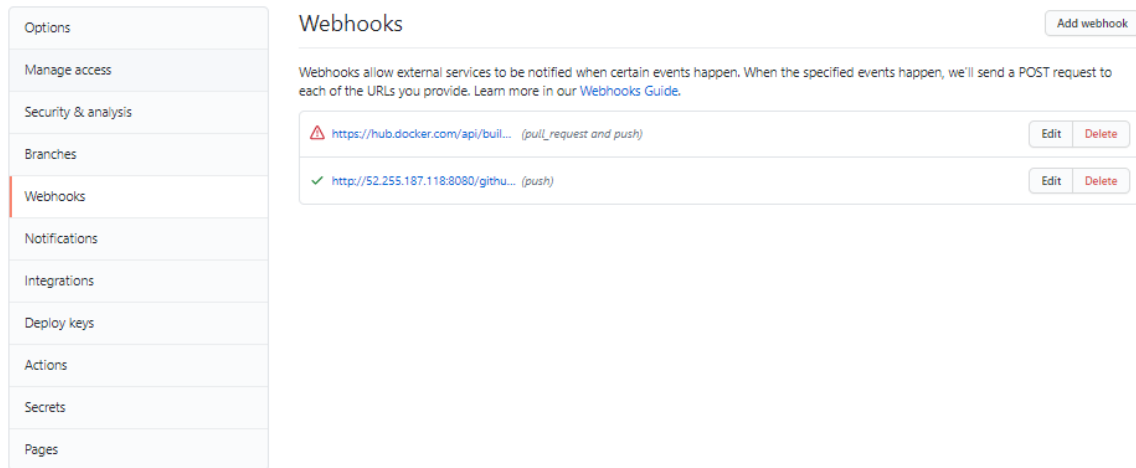
⇒ username에는 github ID 를 입력한다.

⇒ ID (식별자), Description (설명)은 마음대로 작성한다.

7. github 프로젝트의 Settings에 접속



8. Webhooks 클릭후 Add webhook 클릭



9. Webhooks 등록, 다음과 같이 설정한다.

Webhooks / Manage webhook

Settings Recent Deliveries

We'll send a post request to the URL below with details of any subscribed events. You can also specify which data format you'd like to receive (JSON, x-www-form-urlencoded, etc). More information can be found in [our developer documentation](#).

Payload URL *

Content type

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me everything.

☐ Let me select individual events.

☒ **Active**
We will deliver event details when this hook is triggered.

⇒ Payload URL : `http://jenkins_server_ip:portnumber/github-webhook/` 를 붙여서 등록한다.

⇒ Content type : `application/json`

⇒ 푸시할때 jenkins가 빌드하도록 할 경우 Just the push event. 체크

⇒ Active 체크 후 저장

10. Jenkins 프로젝트 설정에 Build Triggers의 GitHub hook trigger for GITScm polling을 체크한다.

Build Triggers

☐ Build after other projects are built

☐ Build periodically

☐ GitHub Branches

☐ GitHub Pull Requests

☒ **GitHub hook trigger for GITScm polling**

☐ Poll SCM

☐ 빌드 안함

☐ Quiet period

☐ 빌드를 원격으로 유발 (예: 스크립트 사용)

AWS EC2 등록(SSH Username with private key)

Kind

SSH Username with private key

Scope

Global (Jenkins, nodes, items, all child items, etc)

ID

ec2-server2

Description

AWS EC2 서버

Username

ubuntu (ec2의 username과 동일해야함, 중요!!!!)

☐ Treat username as secret

Private Key

☒ Enter directly

Key

aws ec2 pem키를 여기에 복붙한다.

Enter New Secret Below

1. 위의 사진 설명대로 입력한다.

⇒ **ec2 서버의 사용자 이름과 Username이 일치하지 않을 경우 정상적으로 동작하지 않음!!!**

→ ec2를 ubuntu로 한 경우에는 ubuntu가 디폴트

→ Amazon Linux일 경우 ec2-user 가 디폴트

⇒ ID는 파이프라인에서 ssh를 사용하기 위한 식별자, 기억하기 쉬운거로 작성을 권장한다.

2. 플러그인 설치

→ **SSH Agent plugin**

단순히 원격 서버에 명령어만 실행하고자 할 경우 사용, 검색하면서 느낀건 그렇게 많이 사용하는 플러그인 같지는 않다.

도커로 배포할 경우에는 별도 파일 전송이 필요없이 도커 명령어만으로 배포가 가능하기 때문에 꽤 유용하다.

→ **Publish over SSH**

원격 서버에 파일을 전송하고 명령어도 실행하고자 할 경우 사용, 많이 사용되는 플러그인.

압축파일이나 빌드된 파일을 전송한 다음 실행할 때 유용하다.

3. 파이프라인 작성

```
pipeline {
    agent any

    stages {
        stage('git pull') {
            steps {
                echo '-----git Clone Repository-----'
                git credentialsId: 'sjabber', url: 'https://github.com/sjabber/demojenkins'
            }
        }

        post {
            failure {
                echo 'git pull fail...'
            }
        }
    }
}
```

```

    }
  }
}

stage('build') {
  steps {
    echo "-----java build start-----"
    sh script: '''
    #!/bin/bash
    cd ./src
    echo "this is $(pwd)"
    javac hello.java
    '''
  }
  post {
    failure {
      echo 'build fail...'
    }
  }
}

stage('transfer') {
  steps {
    sh script: '''
    cd ./src
    touch a.txt
    pwd
    ls -al
    sudo -i
    '''
    sshPublisher(
      //continueOnError => 이전 서버 배포 실패 이후 다른 서버에 배포할 건지 선택
      //failOnError => 서버 계시에 문제가 있을 경우 빌드를 실패로 표시할 것인지. default 는 true
      continueOnError: false, failOnError: true,
      publishers: [
        sshPublisherDesc(
          configName: "ec2-server", //Jenkins 시스템 정보(Manage Credentials)에 사전 입력한 서버 ID
          verbose: true,
          transfers: [
            sshTransfer(
              // 전송할 파일
              // 주의!!!! jenkins 워크스페이스상의 상대경로로 입력해야한다!!
              // 내 jenkins 서버의 절대경로 : /var/jenkins_home/workspace/test
              sourceFiles: "src/hello.class",
              // sourceFiles: "test/src/hello.class",

              //파일에서 삭제할 경로가 있다면 작성, 제거할 접두사를 의미한다.
              // EX) 작업공간 A/B/C/test.jar 가 있을 때 C/test.jar로 배포하고 싶다면
              //     여기에 A/B를 입력해주면 된다.
              removePrefix: "src",

              //배포할 위치
              //배포할 디렉토리의 게스트 권한 w 활성화!!
              //chmod 747 ./practice 로 설정했음.
              remoteDirectory: "/practice/",

              //원격지에서 실행할 커맨드
              // 바로 실행하려면 여기에 커맨드를 적어주어도 된다.
              execCommand: ""
            )
          ]
        )
      ]
    )
  }
  post {
    failure {
      echo 'transfer fail...'
    }
  }
}
}

```

```

stage('run') {
    steps {
        echo '-----run java-----'
        sshagent(['ec2-server']) {
            sh '''
                ssh -o StrictHostKeyChecking=no ubuntu@15.165.17.133 sh hello_sh
            '''
        }
    }
    post {
        failure {
            echo 'java run fail...'
        }
    }
}
}
}

```