

* Jenkins로 CI/CD 적용한 이야기

- 발단: 업데이트할 때마다 새롭게 배포하기 너무 귀찮다!!!
- Docker로 jenkins 설치하고 실행하기
 - jenkins 이미지 다운로드
 - jenkins 컨테이너 실행
 - jenkins 컨테이너 작동 확인
 - jenkins 웹 페이지 초기 세팅
- CI/CD를 적용할 아이템 등록
- Gitlab과 연결을 위한 플러그인 설치
 - SSH 플러그인 설치 및 환경 설정
 - Gitlab 플러그인 설치 및 환경 설정
 - 소스코드 관리 설정하기
 - 빌드 유발 설정하기
 - Gitlab 시크릿 토큰값 설정하기
- Webhook 연결하기
- 빌드 플러그인 설정
 - 빌드를 위한 플러그인 설치
 - 빌드 플러그인 설정하기
 - 빌드 환경 설정하기
 - Pull 받아오기 테스트
- 빌드하기
 - 빌드할 내용 입력하기
 - 빌드 후 조치하기
- 끝!! CI/CD 작동 확인하기
- 후기

만든이



발단: 업데이트할 때마다 새롭게 배포하기 너무 귀찮다!!!



웹을 개발하면서 매번 업데이트가 생길 때마다 배포하기가 너무 귀찮아요!! 업데이트가 생길 때마다 pull 땡기고, 빌드하고, 디플로이까지... 단순 반복작업아닌가? 생각이 들 때가 많죠. 깃랩 레파지토리에 push 이벤트를 감지하고 알아서 pull하고, 빌드와 배포까지 척척!! 하는 CI/CD 를 적용해보려고 합니다.

현재 싸피 깃랩 내부 jenkins를 이용하여 CD를 적용하는 것은 까다롭습니다. 내부 jenkins에서는 각 빌드 툴 버전에 맞는 플러그인을 유연하게 설치할 수 없어요. 여러 가지 해결방안이 있지만 절 차상 까다롭거나 보안적으로 위험해요... 또르륵

그래서 jenkins를 AWS EC2 인스턴스에 설치해서 사용해서 적용해보도록 하겠습니다.

1 Docker로 jenkins 설치하고 실행하기

jenkins 이미지 다운로드

도커를 이용해서 jenkins를 설치하는 것이 너무나도 쉽고 간편하기 때문에, 도커를 이용하여 jenkins 설치를 해보겠습니다. (jenkins를 어떻게 설치하든 상관이 없어요!)

```
$ sudo docker pull jenkins/jenkins:lts

lts: Pulling from jenkins/jenkins
3192219af04: Pull complete
17c160265e75: Pull complete
cc4fe40d0e61: Pull complete
9d647f502a07: Pull complete
d108b8c498aa: Pull complete
1bfe918b8aa5: Pull complete
dafa1a7c0751: Pull complete
9221a8ef4852: Pull complete
a79e75dd432b: Pull complete
efbc20726efc: Pull complete
50b961ecb92e: Pull complete
8286df7b79f8: Pull complete
a346c2730790: Pull complete
603e97483b95: Pull complete
bdea87e88ba5: Pull complete
5e7cc75a7564: Pull complete
236882534dd3: Pull complete
f16f84daa8d5: Pull complete
```

```
cf2c274807b: Pull complete  
Digest: sha256:971abc4c5db4ebc024a501236f7054bbf480d67a622bb13b453de71f37f91d6b  
Status: Downloaded newer image for jenkins/jenkins:lts
```

jenkins 컨테이너 실행

이제 다운 받은 jenkins 이미지를 실행해줍니다.

```
$ sudo docker run -d -p 8080:8080 -v /app/swim:/var/jenkins_home --name swim_jenkins -u root jenkins/jenkins:lts
```

-d 백그라운드로 실행

-p 호스트 8080포트와 도커 네트워크 상의 8080포트를 연결

-v 호스트의 파일 시스템과 도커 컨테이너 파일 시스템 연결(/app/swim 디렉터리에 /var/jekins_home을 마운트시킨다)

--name 도커 컨테이너 이름 지정 (여기서는 swim_jenkins)

-u 사용자를 root로 지정

jenkins컨테이너 작동 확인

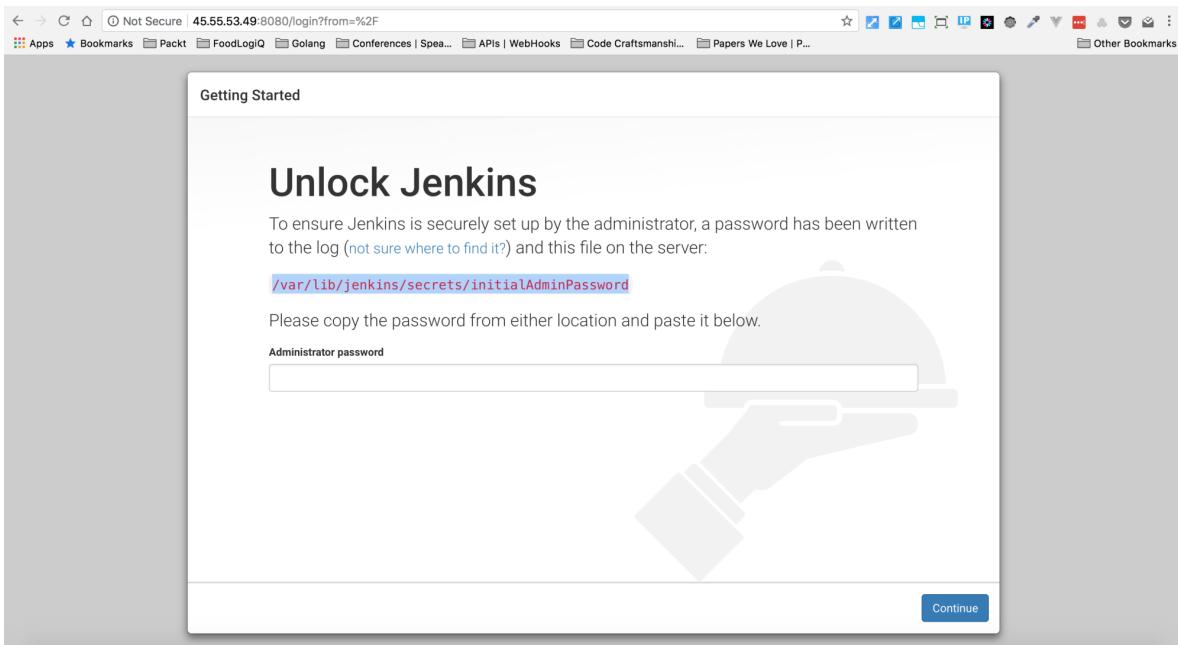
도커 컨테이너가 정상적으로 작동하는지 확인해봅니다.

```
$ sudo docker ps  
CONTAINER ID        IMAGE               COMMAND  
a376e39794c7        jenkins/jenkins:lts   "/sbin/tini -- /usr/..."  
  
CREATED              STATUS  
42 hours ago        Up 42 hours  
  
PORTS                NAMES  
0.0.0.0:8080->8080/tcp, 50000/tcp    swim_jenkins
```

컨테이너 실행할 때, 넣어준 옵션으로 동작하는 것을 확인할 수 있습니다!!

jenkins 웹 페이지 초기 세팅

http://{HOST_NAME}:8080 으로 접근해서 jenkins 설정 웹페이지에 접근합니다.



```
$ sudo docker exec -it swim_jenkins bash
root@a376e39794c7:/# cat /var/lib/secrets/initialAdminPassword
asdjhaskdklqhk123asd
root@a376e39794c7:/# exit
```

컨테이너에 접근해서 비밀번호를 알아온 후에 빈 칸에 넣어줍니다. 그러면 이제 본격적으로 jenkins를 사용할 수 있는 준비가 끝났습니다!!

2 CI/CD를 적용할 아이템 등록

jenkins 에 들어가서 CI/CD를 적용할 아이템 등록합니다.

The screenshot shows the Jenkins dashboard. The left sidebar has a dark background with white icons and text. The "새로운 Item" option is highlighted with a blue rectangular box. Other items in the sidebar include "사람", "빌드 기록", "프로젝트 연관 관계", "파일 핑거프린트 확인", "Jenkins 관리", "My Views", "Lockable Resources", and "New View". The top navigation bar is black with white text, showing the Jenkins logo and the word "Jenkins". The search bar and other user interface elements are also visible.

Freestyle project에 studypro라는 이름으로 만들어봤습니다.

The screenshot shows the Jenkins interface with a search bar at the top containing the text 'studypro'. Below the search bar, there is a required field indicator '» Required field'. A 'Freestyle project' card is displayed, featuring an icon of a blue folder with a white gear, the text 'Freestyle project', and a description: '이것은 Jenkins의 주요 기능입니다. Jenkins은 어느 빌드 시스템과 어떤 SCM(형상관리)으로 묶인 당신의 프로젝트를 빌드할 것이고, 소프트웨어 빌드보다 다른 어떤 것에 자주 사용될 수 있습니다.' Below it, a 'Pipeline' card is shown with an icon of a blue gear, the text 'Pipeline', and a description: 'Orchestrates long-running activities that can span multiple build agents. Suitable for building pipelines (formerly known as workflows) and/or organizing complex activities that do not easily fit in free-style job type.' At the bottom of the dialog, there is a 'Multi-configuration project' card with an icon of a blue gear and a document, followed by a large blue 'OK' button.

3 Gitlab과 연결을 위한 플러그인 설치

SSH 플러그인 설치 및 설정

Jenkins 관리 > 플러그인 관리 > 설치 가능에 들어가서 Publish Over SSH 설치합니다.

The screenshot shows the Jenkins 'Publish Over SSH' configuration page. It includes a note about using JSON with the ssn-credentials plugin. The 'Send build artifacts over SSH' checkbox is checked. A message box states: 'This plugin is up for adoption! We are looking for new maintainers. Visit our Adopt a Plugin initiative for more information.' The version is listed as 1.20.1, and there is a '설치 제거' (Uninstall) button.

Jenkins 관리 > 시스템 설정 > Publish over SSH 설정에 들어가서 AWS 접근에 필요한 key 파일을 지정해줍니다.

1. Key 파일 지정

test configuration by sending test e-mail

Publish over SSH

Jenkins SSH Key

Passphrase	<input type="text" value="Concealed"/>	?
Path to key	<input type="text"/>	?
Key	<pre>-----BEGIN RSA PRIVATE KEY----- [REDACTED] -----END RSA PRIVATE KEY-----</pre>	

접근하려고 하는 AWS의 hostname과 username을 등록해줍니다.

2. ssh server 설정

SSH Servers	<table border="1"><tr><td>SSH Server</td><td>Name: ssafy_coach_14</td></tr><tr><td>Hostname</td><td><input type="text" value="REDACTED"/></td></tr><tr><td>Username</td><td>ubuntu</td></tr><tr><td>Remote Directory</td><td><input type="text"/></td></tr></table>	SSH Server	Name: ssafy_coach_14	Hostname	<input type="text" value="REDACTED"/>	Username	ubuntu	Remote Directory	<input type="text"/>	고급...
SSH Server	Name: ssafy_coach_14									
Hostname	<input type="text" value="REDACTED"/>									
Username	ubuntu									
Remote Directory	<input type="text"/>									
		Test Configuration								
		삭제								
제공받은 hostname과 username을 적어주면 됩니다. 예시) Hostname: i3a110.p.ssafy.io Username: ubuntu										
추가										

Gitlab 플러그인 설치 및 설정

jenkins 관리 > 플러그인 관리 > 설치 가능에 들어가서 GitLab Plugin 설치합니다.

This plugin integrates Git with Jenkins.	4.3.0	설치 제거
GitLab Plugin		
<input checked="" type="checkbox"/> This plugin allows GitLab to trigger Jenkins builds and display their results in the GitLab UI.	1.5.13	설치 제거
<input checked="" type="checkbox"/> Matrix Project Plugin	1.17	설치 제거

jenkins 관리 > 시스템 설정 > gitlab 설정에 들어가서 접근하려고 하는 깃랩을 지정해줍니다.

List of default health metrics for Folders

Gitlab

Enable authentication for '/project' end-point

GitLab connections

Connection name	studypro
A name for the connection	
Gitlab host URL	https://lab.ssafy.com
Credentials	- none - <input type="button" value="Add"/>
API Token for Gitlab access required	
API Token for accessing Gitlab	
<input type="button" value="고급..."/>	
<input type="button" value="Test Connection"/>	
<input type="button" value="삭제"/>	

`credentials`가 없다면 Add를 눌러 새롭게 추가합니다. 저는 GitLab API token을 이용해서 등록했습니다.

Gitlab API는 로그인한 다음에 `User settings > Access Tokens`에 들어가면 받을 수 있습니다.

Jenkins Credentials Provider: Jenkins

Add Credentials

Domain	Global credentials (unrestricted)
Kind	GitLab API token
Scope	Global (Jenkins, nodes, items, all child items, etc)
API token	각자 발급받은 API token을 지정해줍니다
ID	swim_studypro_id
Description	gitlab connection with API token
<input type="button" value="Add"/> <input type="button" value="Cancel"/>	

그리고 이제 등록해줍니다.

List of default health metrics for Folders

Gitlab

Enable authentication for '/project' end-point

GitLab connections

Connection name	studypro
A name for the connection	
Gitlab host URL	https://lab.ssafy.com
Credentials	GitLab API token (gitlab connection with API token) <input type="button" value="Add"/>
API Token for accessing Gitlab	
<input type="button" value="고급..."/>	
<input type="button" value="Test Connection"/>	
<input type="button" value="삭제"/>	

Gitlab 플러그인 connection 확인해보기

Test Connection을 눌러서 과연 정말로...!! 깃랩과 연동이 되었는지 확인해봅니다.

List of default health metrics for Folders

Gitlab

Enable authentication for '/project' end-point

GitLab connections

Connection name	studypro
A name for the connection	
Gitlab host URL	https://lab.ssafy.com
Credentials	GitLab API token (gitlab connection with gitlab <input type="button" value="Add"/>

API Token for accessing Gitlab

Success

깃랩과 연동 확인 완료

Gitlab API 토큰을 이용해서 연결이 된 것이 확인됩니다.

소스 코드 관리 설정하기

pull 맹겨올 레파지토리를 등록합니다. 원하는 브랜치도 설정할 수 있어요! 저는 develop 브랜치를 빌드하는데 사용하였습니다. 깃 브랜치 전략에 따라 배포하고 싶은 브랜치로 설정합니다.

General 소스 코드 관리 빌드 유발 빌드 환경 Build 빌드 후 조치

소스 코드 관리

None Git

레포지터리 URL 등록

Repositories Repository URL Credentials ssafy coach 14/******** (sdf)

자격 조건 등록

Branches to build Branch Specifier (blank for 'any') Add Repository

배포하고 싶은 브랜치 지정

Branch Specifier (blank for 'any') Add Branch

Repository browser (자동)

Additional Behaviours

Subversion

빌드 유발 설정하기

webhook 시그널을 받고 빌드할 수 있도록 트리거 설정을 해줍니다.

- Build when a change is pushed to GitLab webhook URL : 을 체크합니다.

The screenshot shows the Jenkins configuration interface for a job named '빌드 유발'. The 'Build' tab is selected. Under the 'Build Triggers' section, the 'Build when a change is pushed to GitLab' checkbox is checked, with a note indicating the GitLab webhook URL: [http://\[REDACTED\]:8080/project/studypro](http://[REDACTED]:8080/project/studypro). Below this, under 'Enabled GitLab triggers', several options are listed with checkboxes: Push Events (checked), Opened Merge Request Events (checked), Accepted Merge Request Events (unchecked), Closed Merge Request Events (unchecked), Rebuild open Merge Requests (set to 'Never'), Approved Merge Requests (EE-only) (checked), and Comments (checked). A comment regex field contains 'Jenkins please retry a build'. A '고급...' button is also visible.

Gitlab 시크릿 토큰값 설정하기

빌드 유발에서 고급 설정에서 시크릿 토큰값을 생성합니다. 이 토큰 값은 GitLab webhook 설정에 필요합니다.

The screenshot shows the Jenkins 'Global Tools Configuration' page. Under the 'Secret Token' section, a text input field contains a long, randomly generated string of characters. This field is highlighted with a large red rectangular box. To the right of the input field are two buttons: 'Generate' (in blue) and 'Clear' (in grey). Other sections visible include 'Ignore WIP Merge Requests' (checked), 'Set build description to build cause (eg. Merge request or Git Push)' (checked), 'Build on successful pipeline events' (unchecked), 'Pending build name for pipeline' (empty input field), 'Cancel pending merge request builds on update' (unchecked), 'Allowed branches' (radio buttons for 'Allow all branches to trigger this job' (selected), 'Filter branches by name', 'Filter branches by regex', and 'Filter merge request by label' (unchecked)), and 'GitHub hook trigger for GITScm polling' and 'Poll SCM' (both unchecked).

4 Webhook 지정하기

gitlab에 pushevent가 생기면 jenkins로 시그널을 보내줘야 합니다. 자동으로 push 이벤트를 감지하고 신호를 보낼 수 있도록 다음과 같이 gitlab webhook 을 지정해줍니다.

gitlab 로그인 후 > setting > integrations

The screenshot shows the 'Integrations' settings page in GitLab. On the left, there's a sidebar with '설습코치 > studypro > Integrations'. The main area has two input fields: 'URL' containing 'http://127.0.0.1:1111/jenkins:8080/project/studypro' and 'Secret Token' containing a redacted string. A large watermark 'URL과 Secret token 입력!' is overlaid on the right side of the form. Below the tokens, a note says 'Use this token to validate received payloads. It will be sent with the request in the X-Gitlab-Token HTTP header.' Under the 'Trigger' section, the 'Push events' checkbox is checked, with a note 'This URL will be triggered by a push to the repository' and a field for 'Branch name or wildcard pattern to trigger on (leave blank for all)'. There are several other unchecked checkboxes for 'Tag push events', 'Comments', 'Confidential Comments', 'Issues events', and 'Confidential Issues events', each with their respective notes.

jenkins에 webhook 시그널이 갈 수 있도록 URL을 입력해줍니다. 빌드 유발에서 체크할 때 명시되는 URL을 적어줍니다.

그리고 인증을 위해 빌드 유발 고급 옵션에서 생성한 시크릿 토큰을 입력해줍니다.

5 빌드 플러그인 설정

빌드를 위한 플러그인 설치

NodeJS 빌드를 위한 NodeJS 플러그인을 설치합니다.

The screenshot shows the Jenkins 'Manage Plugins' page. In the search bar at the top, 'nodejs' is typed. Below the search bar are tabs: '업데이트된 플러그인 목록', '설치 가능', '설치된 플러그인 목록' (which is selected), and '고급'. The main area lists three plugins: 'Config File Provider Plugin' (version 3.6.3), 'NodeJS Plugin' (version 1.3.7), and 'Structs Plugin' (version 1.20). Each plugin has a '설치' (Install) button to its right. The 'NodeJS Plugin' is currently selected, as indicated by a red border around its row.

자바 파일을 빌드하시는 분은 maven 플러그인을 설치합니다.

The screenshot shows the Jenkins Marketplace search results for 'maven'. A red arrow points to the 'Maven Integration' plugin by Jira. The plugin is categorized under 'Build Tools'. It has a version of 3.7 and was released 29 days ago. The description states: 'This plug-in provides, for better and for worse, a deep integration of Jenkins and Maven: Automatic triggers between projects depending on SNAPSHOTs, automated configuration of various Jenkins publishers (JUnit, ...).'

빌드 플러그인 설정하기

jenkins 관리 > Global Tool Configuration에 들어가서 nodejs와 maven 버전을 설치해줍니다. nodejs는 LTS 버전인 12.18.3을 설치했어요. maven은 3.6.3 버전을 설치했습니다.

The screenshot shows the Jenkins Global Tool Configuration page. It is divided into two sections: NodeJS and Maven.

NodeJS section:

- NodeJS installations: A list with one entry named "node 12.18.3".
- Add NodeJS button.
- Install automatically checkbox (checked).
- Install from nodejs.org section:
 - Version: NodeJS 12.18.3 (dropdown menu).
 - Force 32bit architecture checkbox (unchecked).
 - Global npm packages to install input field (empty).
 - Global npm packages refresh hours input field (72).
- Delete Installer button.
- Add Installer dropdown.
- Add NodeJS button.

Maven section:

- Maven installations: A list with one entry named "mvn 3.6.3".
- Add Maven button.
- Install automatically checkbox (checked).
- Install from Apache section:
 - Version: 3.6.3 (dropdown menu).
- Delete Installer button.
- Add Installer dropdown.
- Add Maven button.

List of Maven installations on this system: List of Maven installations on this system

빌드 환경 설정하기

jenkins > [Item_name] > 구성에서 빌드환경을 추가해줍니다.

아까 빌드 플러그인 설치한것을 등록합니다.

빌드 환경

- Delete workspace before build starts
- Use secret text(s) or file(s)
- Provide Configuration files
- Send files or execute commands over SSH before the build starts
- Send files or execute commands over SSH after the build runs
- Abort the build if it's stuck
- Add timestamps to the Console Output
- Inspect build log for published Gradle build scans
- Provide Node & npm bin/ folder to PATH

NodeJS Installation: node 12.18.3
Specify needed nodejs installation where npm installed packages will be provided to the PATH

npmrc file: - use system default -

Cache location: Default (~/.npm or %APP DATA%\npm-cache)

With Ant

저장 **Apply**

Pull 받아오기 테스트

Jenkins > [Item_name]에서 Build Now를 클릭하여 빌드가 되는지 확인봅니다.

결과가 정상적으로 실행되고 나면 git 레파지터리에 있는 develop 브랜치를 땡겨온 것을 확인할 수 있습니다.

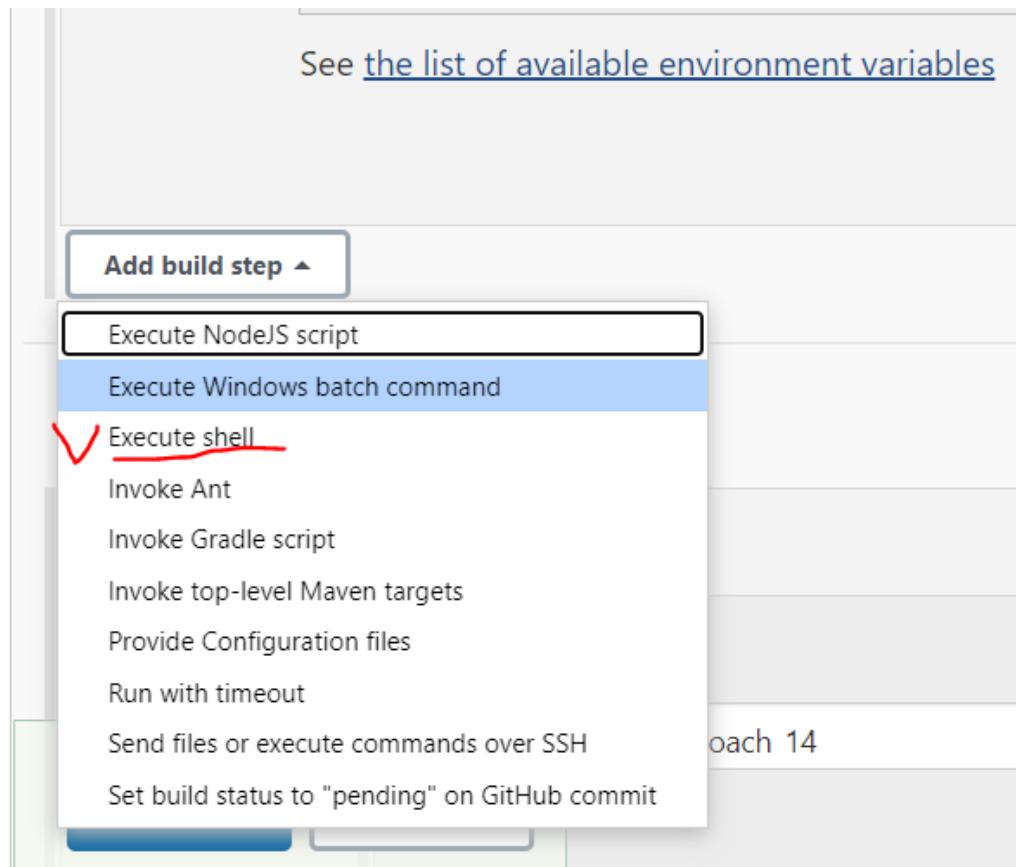
```
ubuntu@ip-172-26-14-51:/app/swim/workspace$ tree studypro -L 1
studypro
├── README.md
├── back
├── front
└── package-lock.json
    └── wireframe
3 directories, 2 files
```

**Docker volume 설정한
/app/swim/workspace에
Git 레파지터리 땡겨오기**

6 빌드하기

빌드할 내용 입력하기

Jenkins > [Item_name] > 구성 > 빌드에서 Execute Shell 항목을 추가합니다.



프론트와 백엔드 각각 나눠서 빌드를 해주었습니다.

The screenshot shows a Jenkins pipeline configuration with two "Execute shell" steps under the "Build" section:

- Execute shell**
Command:

```
cd front
npm install -g yarn
yarn install
yarn build
```
- Execute shell**
Command:

```
cd back
npm install
```

본 예시에서는 Vue와 Express 빌드를 진행했습니다. 각자 개발한 프레임워크에 맞춰서 쉘 명령어를 지정해주면 되요.

1. 프론트 빌드(vue 프론트 배포)

```
cd front
npm install -g yarn
yarn install
yarn build
```

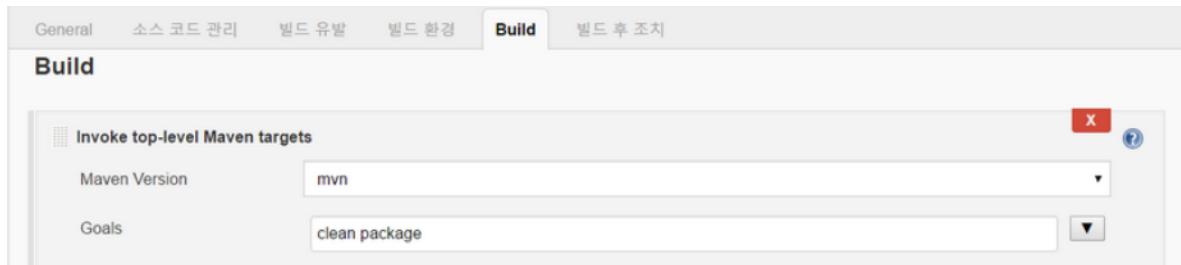
2. 백엔드 빌드 (express 백엔드 배포)

```
cd back  
npm install
```

jar 파일을 빌드하기 하려고 한다면, `invoke top-Level Maven targets` 을 눌러서 배포를 진행하면 됩니다.!

3. 백엔드 빌드 (mvn 백엔드 배포)

`invoke top-Level Maven targets` 항목을 추가해 아래와 같이 작성해주시면 됩니다. Maven Version 명은 각자 다를 수 있겠죠?

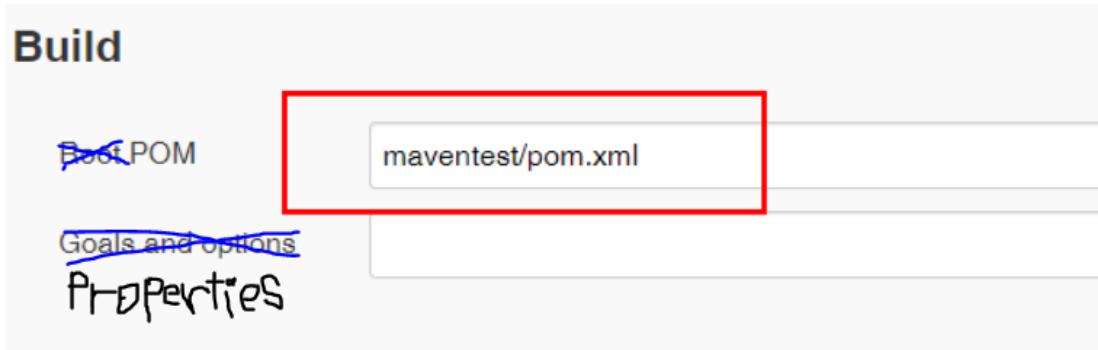


- mvn 백엔드 배포 시 해당 빌드 오류가 생길 경우

콘솔 출력

```
Started by user 관리자  
Building in workspace C:\Program Files (x86)\Jenkins\workspace\TestMavneProject  
Unpacking https://repo.maven.apache.org/maven2/org/apache/maven/apache-maven/3.6.1/apache-maven-3.6.1-bin.zip to C:\Program Files (x86)\Jenkins\tools\hudson.tasks.Maven_MavenInstallation\maven on Jenkins  
Checking out a fresh workspace because there's no workspace at C:\Program Files (x86)\Jenkins\workspace\TestMavneProject  
Cleaning local Directory .  
Checking out https://desktop-2q7h7ok/svn/TestProjectSorce at revision '2019-05-30T14:23:56.470 +0900' --quiet  
Using sole credentials admin/***** in realm '<https://desktop-2q7h7ok:443> VisualSVN Server'  
At revision 3  
  
Parsing POMs  
ERROR: No such file C:\Program Files (x86)\Jenkins\workspace\TestMavneProject\pom.xml  
Perhaps you need to specify the correct POM file path in the project configuration?  
Finished: FAILURE
```

"Invoke top-level Maven targets"에서 "고급" 버튼을 눌러 추가 설정을 펼친 후,



위와 같은 방식으로 POM 파일을 지정해서 하위 디렉토리의 POM 파일을 지정해줘야 합니다. 경로는 각자 다를 수 있겠죠?

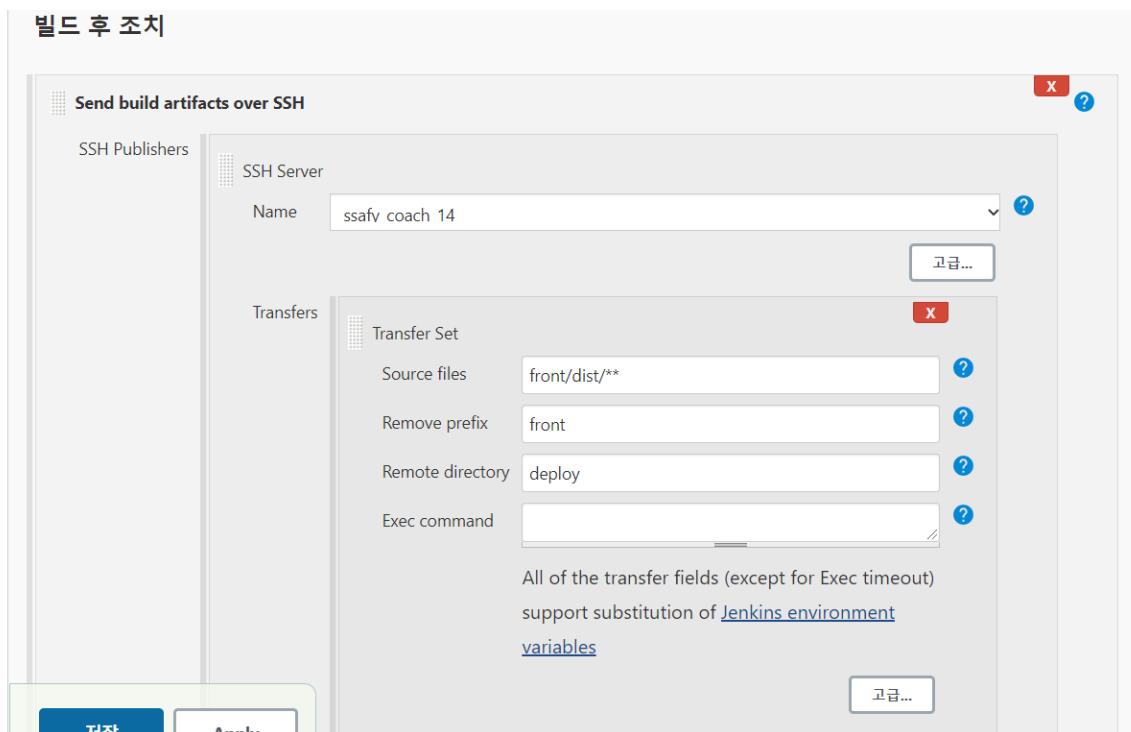
빌드 후 조치하기

Jenkins에서 깃랩 레파지토리 소스 코드를 빌드하고 나서 실행할 명령어를 설정합니다. SSH로 AWS EC2에 접근해서 빌드된 파일을 지정한 곳으로 이동하고 배포하겠습니다.

jenkins > [Item_name] > 설정 > 빌드 후 관리

설정 내용

- *SSH Server Name* : Jenkins 시스템 설정에서 등록한 SSH 서버 중 배포할 서버 선택
- *Source files* : 어떤 파일을 배포할 것인지 설정. `front/**`는 front 디렉터리 밑에 파일을 의미하고, `front/***`는 front 디렉터리 및에 폴더와 파일 모두 의미한다. 또한 `*/.jar`는 모든 폴더의 jar로 끝나는 파일을 의미한다.
- *Remove prefix* : 제거할 접두사를 의미하는데, 기본적으로 작업공간/빌드된 파일들로 되어 있다. 예를 들어 작업공간/A/B/C/test.jar가 있을 때 C/test.jar로 배포하고 싶다면 여기에 A/B를 입력해준다.
- *Remote directory* : Jenkins 시스템 설정에서 SSH 설정시 지정한 훈 디렉토리 뒤에 추가로 입력하는 디렉토리인데, 배포할 파일이 저장될 디렉토리를 지정한다. 이 때 없으면 새로 생성해주지 않기 때문에 미리 만들어 져 있어야 한다.
- *Exec command* : 배포 후 실행 할 명령어를 입력하는 곳으로 리눅스 설정에 따라 특정 명령어를 사용하지 못하므로 설정을 잘 해주거나, 특정 명령어의 절대경로를 입력해주는 방식으로 해결을 해야 한다.



`source files`로 `front/dist/**`을 입력하여 vue build 파일인 dist 디렉터리를 통으로 옮기도록 지정해줍니다.

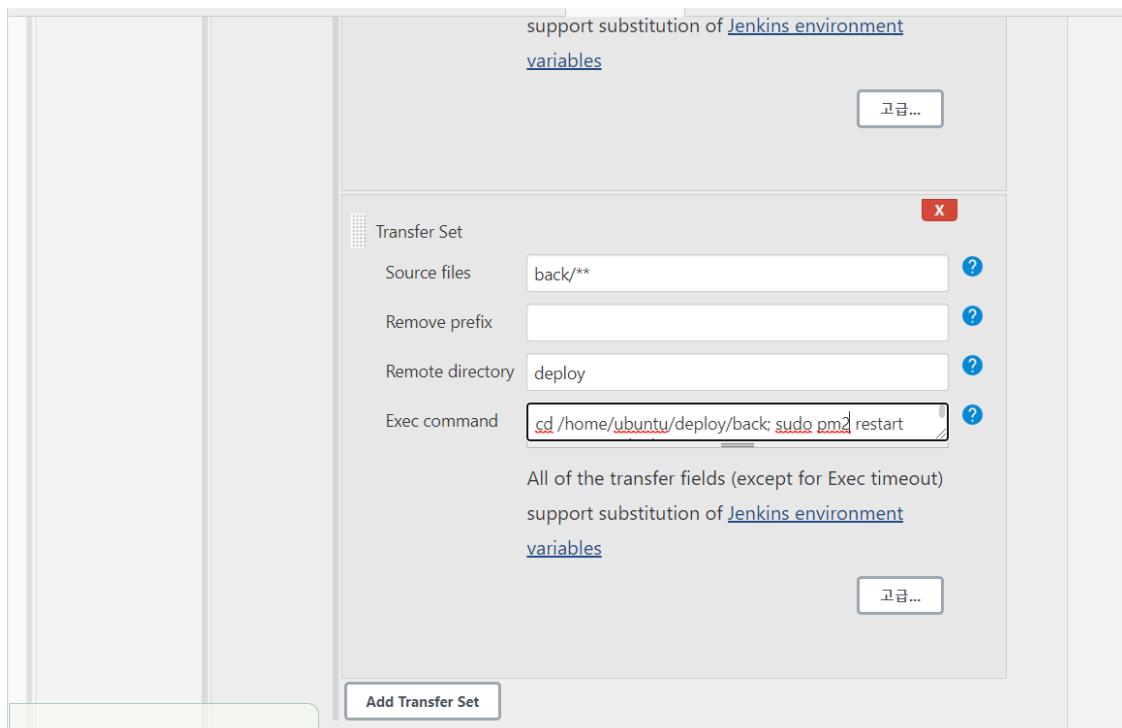
`remove prefix`로 `front`를 지정하여 `remote directory`에 front 디렉터리를 경로 없이 dist 디렉터리가 바로 이동할 수 있도록 지정해줍니다.

위 사항이 반영된 결과는 아래와 같습니다.

```
ubuntu@ip-172-26-14-51:~/deploy/dist$ tree . -L 1
.
├── css
├── favicon.ico
├── fonts
├── img
├── index.html
└── js
    ├── manifest.json
    ├── precache-manifest.cf49b4858001d0d2d89c745dae8e1f9c.js
    ├── robots.txt
    └── service-worker.js

4 directories, 6 files
```

~/deploy/dist 디렉터리 에 빌드파일 이동 확인



마찬가지로 백엔드 파일을 전부 이동시켜준 다음에, pm2 명령어를 이용해서 배포시켜줍니다.

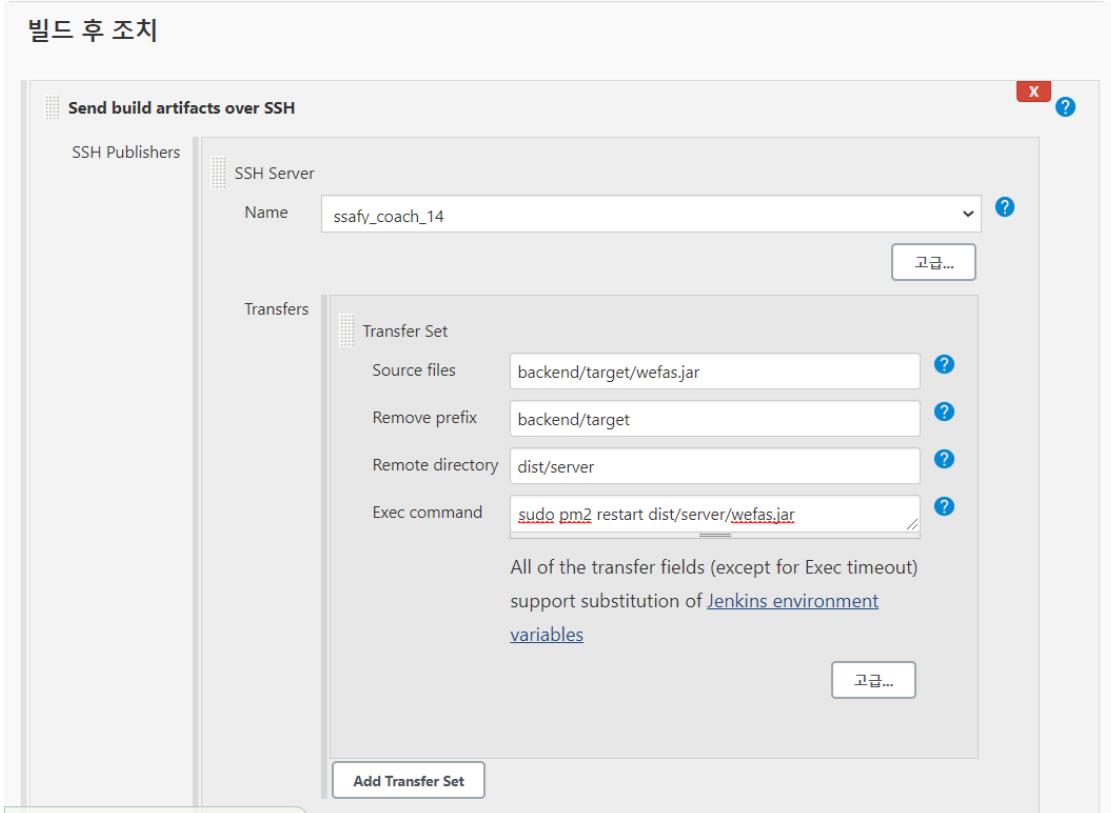
```
ubuntu@ip-172-26-14-51:~/deploy/back$ tree -L 1
.
├── README.md
├── app.js
├── auto_killer
├── config
├── controllers
├── init.js
├── init.sql
├── keys
├── middleware.js
├── models
├── node_modules
├── package-lock.json
├── package.json
├── routers
├── routes.js
├── socket.js
├── stmp.js
└── yarn.lock
```

~/deploy/back 디렉터리에 빌드파일 이동 확인

현재 상태에서 위와 같은 설정은 EC2 내부에서만 파일이 이동하게 됩니다. 그러나 대개 jenkins 서버는 배포 서버와 별도로 두기 때문에, 이를 염두하고 jenkins 서버와 배포 서버가 다른 곳에 있어도 적용할 수 있도록 설정했습니다.

- mvn 배포 시

빌드 후 조치 > send build artifacts over SSH



jar 파일을 이동시켜주고, pm2 명령어를 이용해서 재시작을 해주면 됩니다. 백엔드는 무조건 재 시작을 해주셔야 합니다! Source files, Remote directory 경로는 다를 수 있겠죠!?

- app.json

위의 방법 외에 일반적으로 많이 쓰이는 방법으로 app.json에 경로를 설정해서 `pm2 start app.json`을 통해 재시작 하는 방법이 있습니다. 이 방법으로 할 때 제일 많이 뜨는 에러도 역시 경로 문제입니다. app.json에서 `home/ubuntu/dist/server/jar` 경로로 code를 바꿔주고, 깃에 푸시, 클론을 하셔야 합니다.

7 끝!!! CI/CD 작동 확인하기

커밋하고 푸쉬하면... 짜잔...!! 서버가 자동으로 작동됩니다.

 Jenkins

Jenkins > studypro >

대시보드로 돌아가기

상태

변경사항

작업공간

Build Now

Project 삭제

구성

Rename

Project studypro

jenkins ^.^

작업 공간

최근 변경사항

고정링크

- Last build, (#42), 29 min 전
- Last stable build, (#42), 29 min 전
- Last successful build, (#42), 29 min 전
- Last failed build, (#25), 4 hr 53 min 전
- Last unstable build, (#36), 1 hr 12 min 전
- Last unsuccessful build, (#36), 1 hr 12 min 전
- Last completed build, (#42), 29 min 전

Build History

주요 =

find

#42 2020. 8. 12 오전 6:39

#41 2020. 8. 12 오전 6:35

#40 2020. 8. 12 오전 6:21

#39 2020. 8. 12 오전 6:12

#38 2020. 8. 12 오전 6:12

↳ packages are looking for renaming
run `npm fund` for details

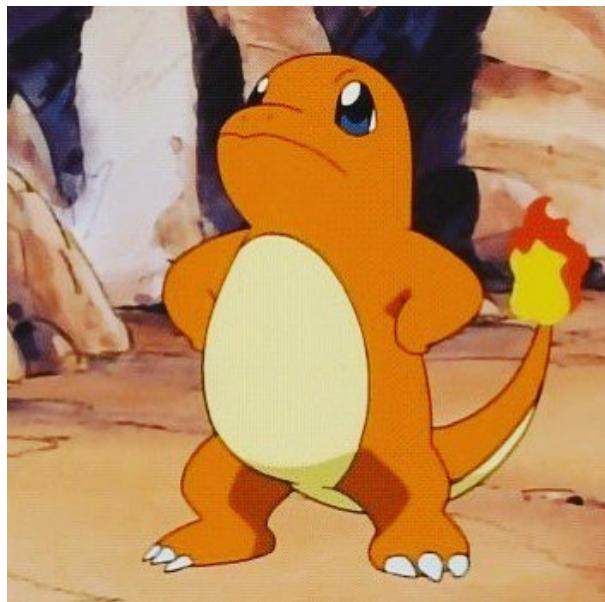
found 5 low severity vulnerabilities
run `npm audit fix` to fix them, or `npm audit` for details

SSH: Connecting from host [a376e39794c7]
SSH: Connecting with configuration [ssafy_coach_14] ...
SSH: EXEC: STDOUT/STDERR from command [cd /home/ubuntu/deploy/back; sudo pm2 restart npm -- run deploy] ...
Use --update-env to update environment variables

```
[PM2] Applying action restartProcessId on app [npm](ids: [ 0 ])
[PM2] [npm](0) ✓
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| id | name   | namespace | version | mode    | pid     | uptime | ⚡ | status  | cpu    | mem
| user | watching |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
| 0  | npm    | default   | N/A     | fork    | 10589   | 0s     | 5  | online  | 0%    | 9.7mb
| root | disabled |
-----+-----+-----+-----+-----+-----+-----+-----+-----+-----+
```

SSH: EXEC: completed after 601 ms
SSH: Disconnecting configuration [ssafy_coach_14] ...
SSH: Transferred 20616 (117 + 20499) file(s)
Finished: SUCCESS

搦 후기



CI/CD를 적용하는 방식은 너무나도 다양합니다. (깃랩 runner을 이용하는 방법도 있고, 도커 컨테이너 잔뜩 올리고 쿠버네틱스 이용하는 방식까지... 엄청 다양합니다!) 이 글은 깃랩에 올린 프로젝트에 간편히 CI/CD 적용하기 위한 방법 중 하나입니다. 각자 상황에 맞춰서 변경해보고, 이것저것 플러그인 설치해보면서 진행하는 것도 좋을 것 같네요. ☺

(END) 후기



우선 자료 다 만들어 주시고, 정말 고생하신 이수영 실습코치님, 좋은 피드백 해주신 황수재 컨설턴트님 너무 감사드립니다!

교육생 여러분! 공통 프로젝트 때 CI/CD를 적용하지 못했다거나, CI/CD 적용하는 것을 맡지 않으신 분들은 다음 프로젝트 때, 아님 혼자서라도 이 자료 보면서 CI/CD 꼭 적용해보셨으면 좋겠습니다. 아마 무조건 따라하기만 하면 잘 안되실 거예요. 여러분들 원격 저장소, 배포 환경에 맞춰 스스로 바꿔보고, CI/CD에 대한 개념도 찾아보면서 시도하시면 좋을 것 같아요.

혹시 관련해서 질문할게 생기시면 이수영 코치님한테 질문해주시고(전 곧 싸피를 떠나기 때문에 ☺), 모두 화이팅입니다!!