

# 팀과제#6 Jenkins (TA)

보고서는 **pdf 형식**으로, 제출 파일들과 함께 압축하여 제출해주세요.  
팀에서 대표로 1명만 제출해주시면 됩니다.

## 평가 기준

- 파일 형식, 파일명을 맞췄는지
  - 최종 제출 파일: 팀과제\_6\_팀번호\_팀명.zip
  - 보고서명: 팀과제\_6\_팀번호\_팀명.pdf
  - 프로젝트 파일/기타 파일명: 자유
- 과제에서 언급한 내용이 모두 명시되어 있고, 내용이 적절한지

위에 언급한 항목에 대해 평가할 예정이며, 연장 제출 받지 않겠습니다.

## 과제 내용

지난 시간에는 Docker Compose를 사용하여 컨테이너 생성을 간략화하는 방법을 알아보았습니다. 이번 시간에는 Jenkins를 사용하여 Docker Compose에서 사용한 방식대로 Build하고 이를 자동화하는 방법을 알아보시다.

### Jenkins 컨테이너 접속

1. Docker hub에서 Jenkins 이미지를 pull받아주세요..

참고: [https://hub.docker.com/\\_/jenkins](https://hub.docker.com/_/jenkins)

```
docker pull jenkins/jenkins:ls
```

```
PS C:\Users\kimmi> docker pull jenkins/jenkins:ls
ls: Pulling from jenkins/jenkins
17c9e6141fdb: Already exists
7a89275704e1: Pull complete
e58eeb56060c: Pull complete
85eb3e033507: Pull complete
5a17900b3cc9: Pull complete
1d84870be6bd: Pull complete
b5439463db64: Pull complete
89bb9dfa5344: Pull complete
7f99eee22e5e: Pull complete
1022f457066c: Pull complete
503c313e2b11: Pull complete
381377c6a31e: Pull complete
0d15342921ba: Pull complete
66a5d1b6ca12: Pull complete
Digest: sha256:4283a1d1ce2f7893d083ec413d1c196edfddc3fe6e0e9ecfefcfbe99bf3bf
f9d
Status: Downloaded newer image for jenkins/jenkins:ls
docker.io/jenkins/jenkins:ls
```

2. pull받은 이미지를 컨테이너로 제작하여 컨테이너를 실행해주세요.

Jenkins는 8080번 포트를 사용하였기 때문에 아래와 같은 명령어를 사용하였습니다.

```
docker run -d -p 8080:8080 -v /home/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock --name {container명} -u
```

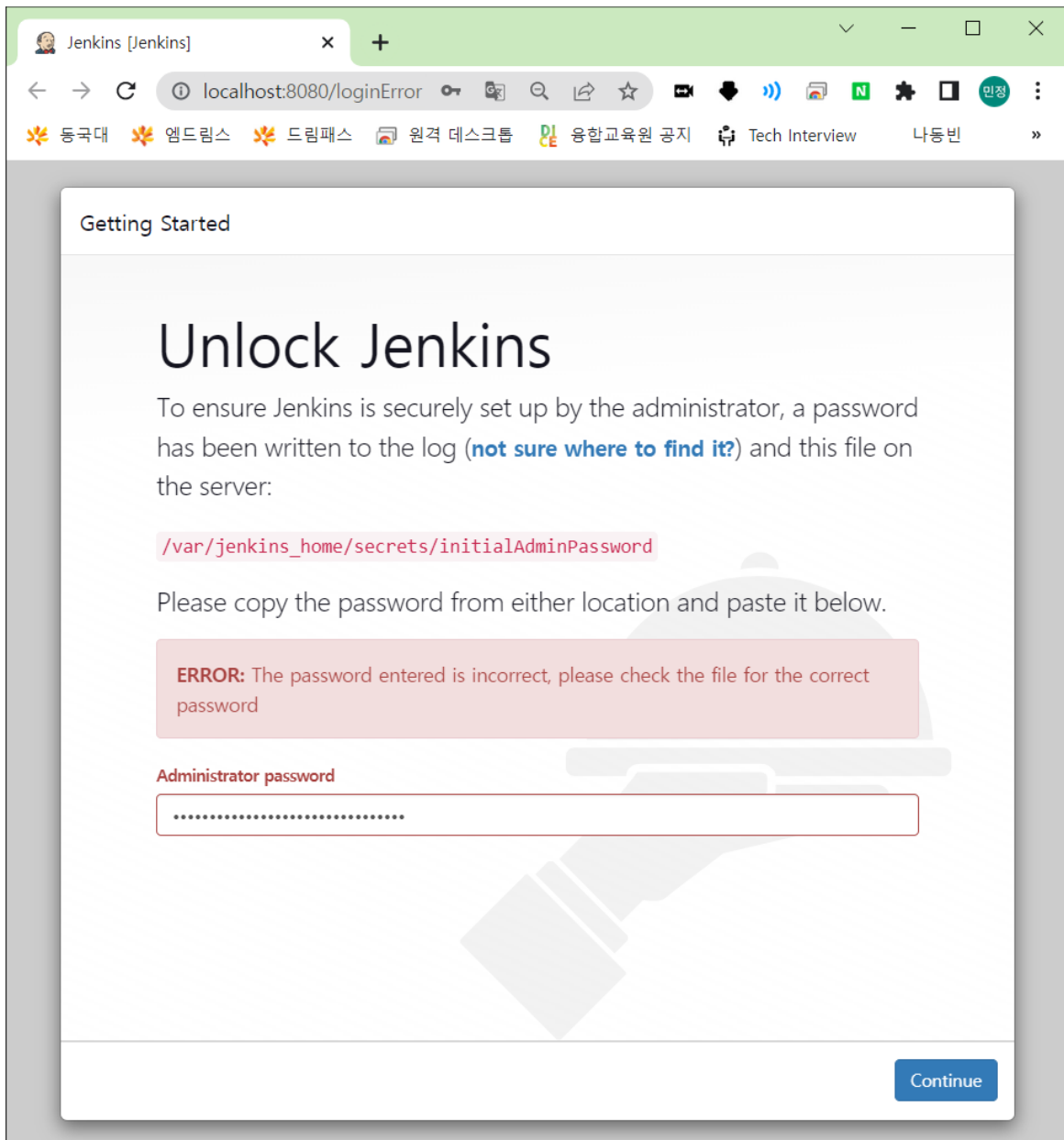
```
PS C:\Users\kimmi> docker run -d -p 8080:8080 -v /home/jenkins:/var/jenkins_
home -v /var/run/docker.sock:/var/run/docker.sock --name hw6 -u root jenkins
/jenkins:lts
87d17359961cbaa5c6862bb4fa397d8bbe7a49320b449e3b5af0bd0e08df0aba
PS C:\Users\kimmi>
```

```
docker exec -it hw6 /bin/bash
```

```
PS C:\Users\kimmi> docker exec -it hw6 /bin/bash
root@87d17359961c:/# |
```

### 3. Jenkins에 접속하세요.

<http://localhost:8080> 에 접속하면, 아래와 같은 화면이 뜹니다.

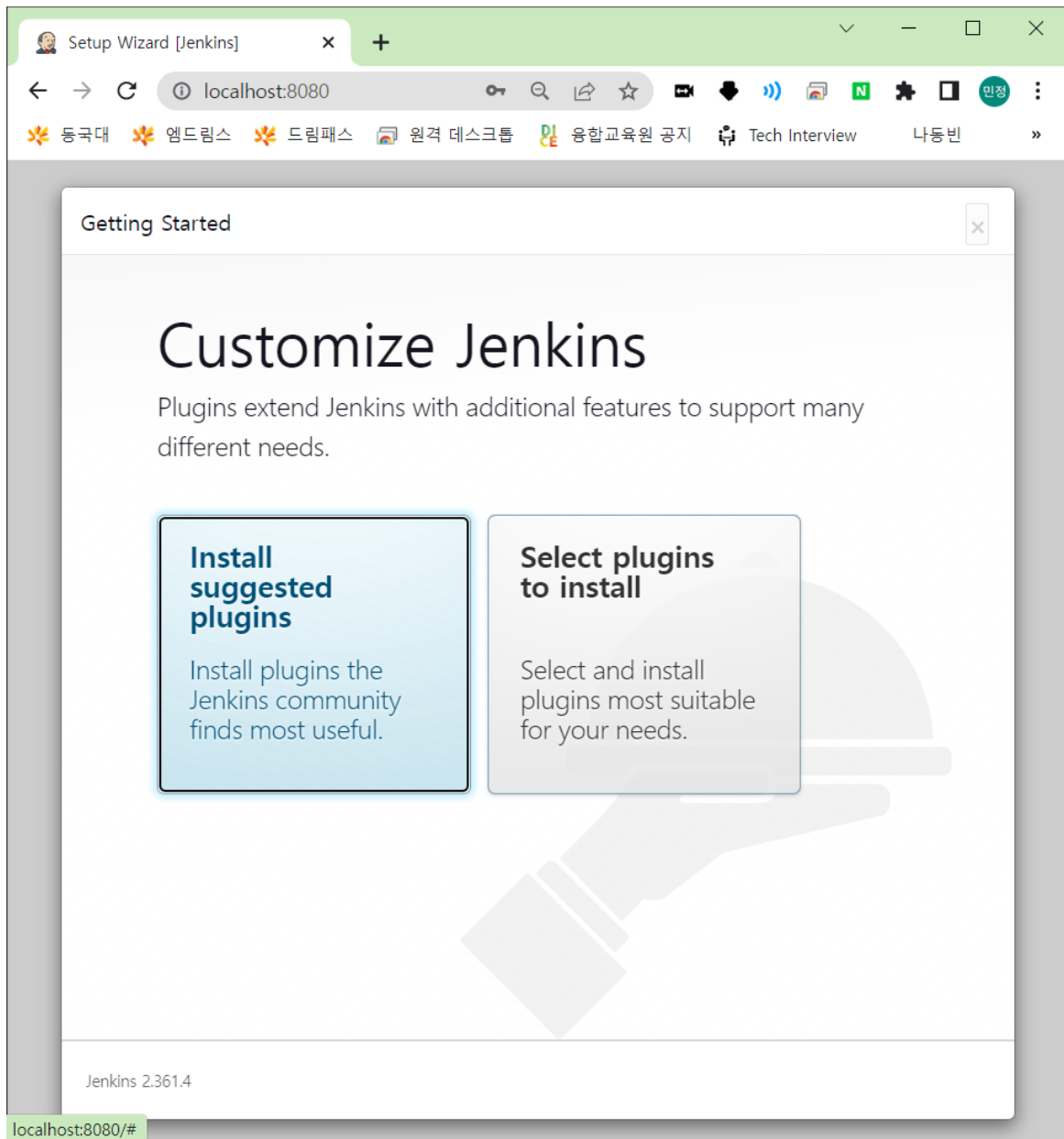


이때, Jenkins container 내에서 아래의 명령어를 사용하여 password를 확인하고 로그인하세요.

```
# container 내에서 실행
cat /var/jenkins_home/secrets/initialAdminPassword
```

```
root@87d17359961c:/# cat /var/jenkins_home/secrets/initialAdminPassword
02370ecdcc1449eea7055b91ade4929e
root@87d17359961c:/#
```

아래와 같은 화면이 뜨면 Install Suggested Plugins를 하시면 됩니다.



#### 4. Admin User를 추가해주세요.

저는 아래와 같이 user를 제작하였습니다.

Setup Wizard [Jenkins]

localhost:8080

Getting Started

## Create First Admin User

계정명:

암호:

암호 확인:

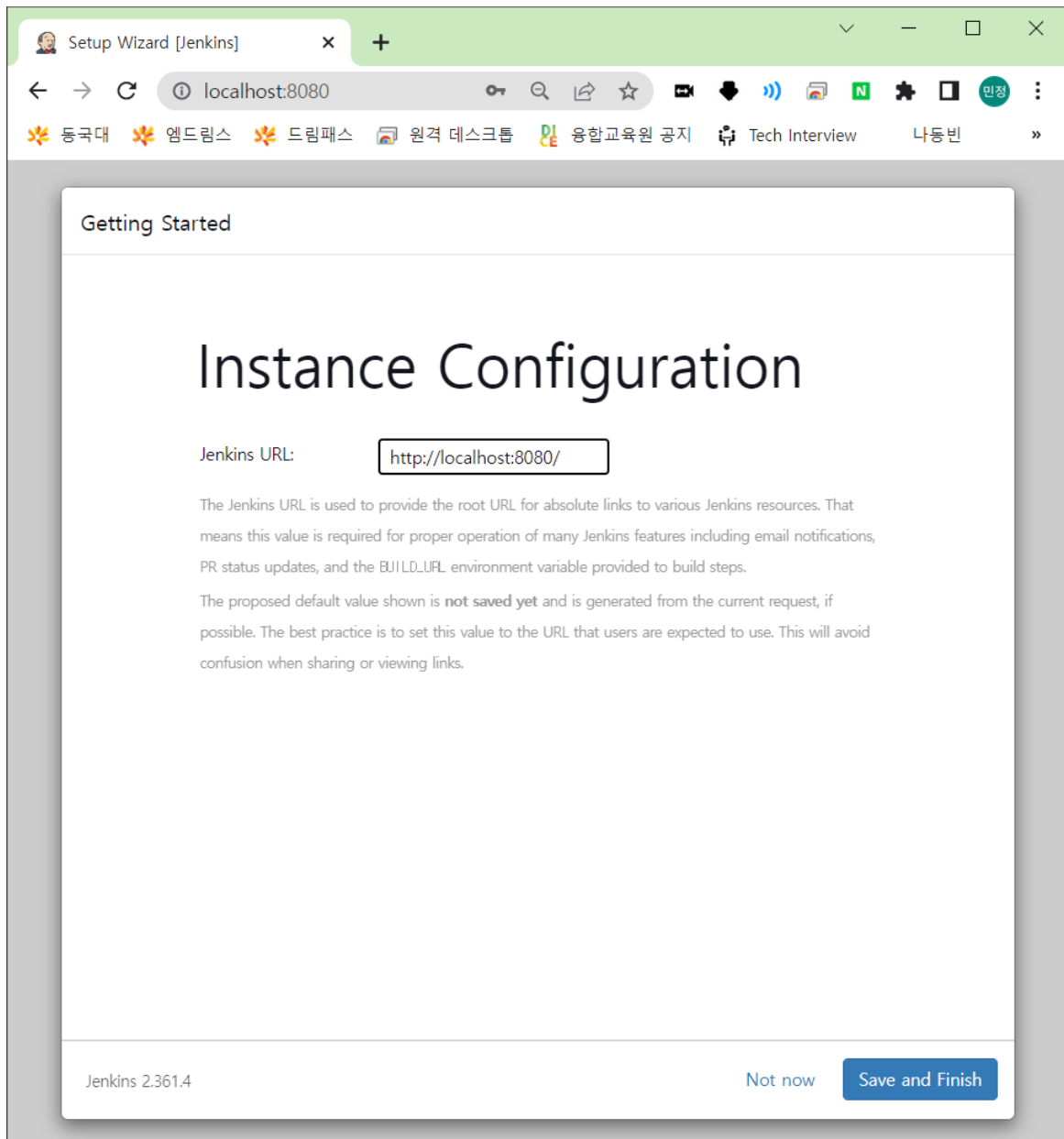
이름:

이메일 주소:

Jenkins 2.361.4

[Skip and continue as admin](#) [Save and Continue](#)

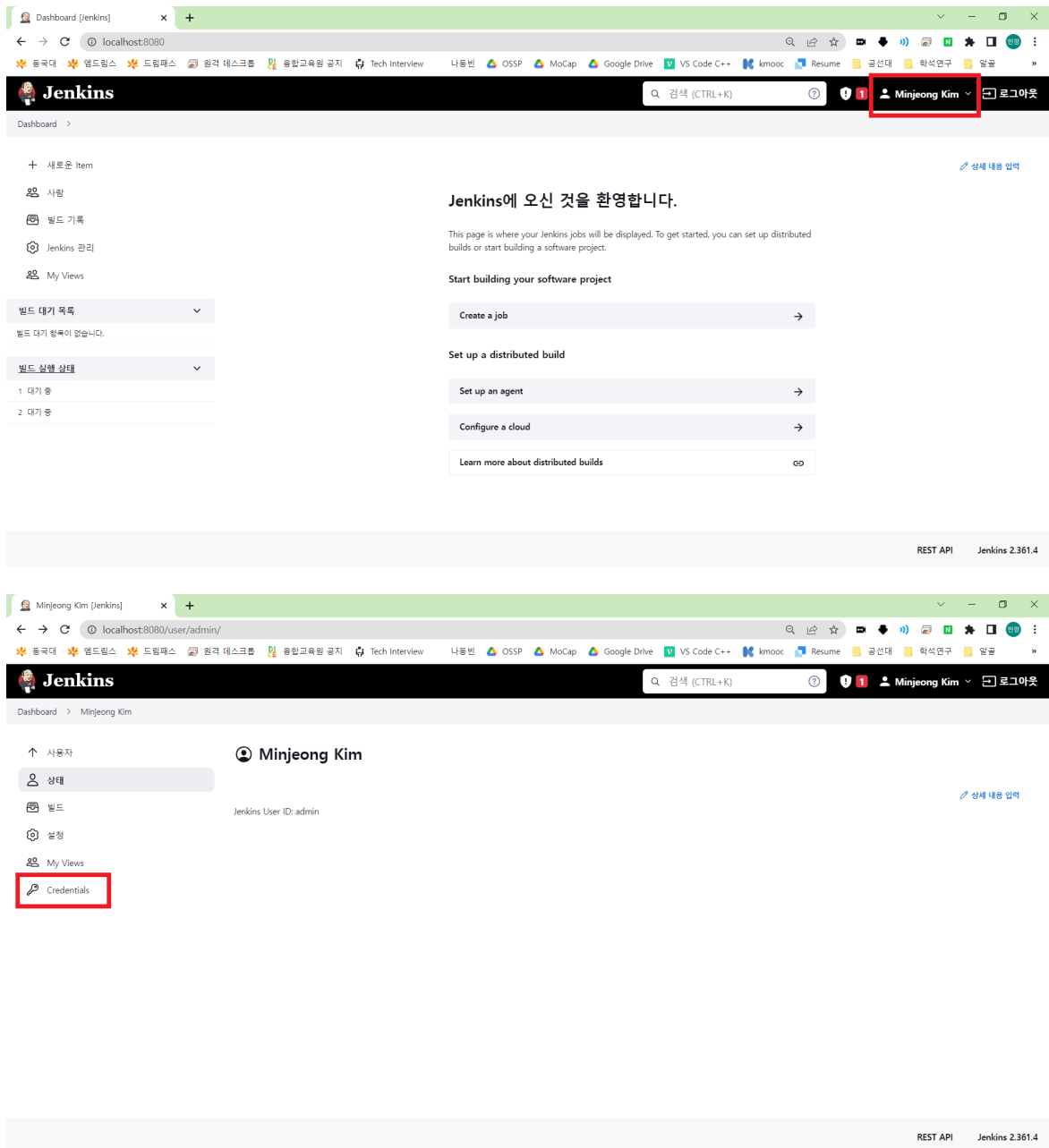
이후 URL을 선택해주세요. 저는 수정 없이 8080 포트를 이용할 예정입니다. 해당 부분은 그대로 이용하시는게 좋습니다.



## Jenkins 작업 추가

### 5. credentials를 추가해주세요.

오른쪽 상단 본인의 이름을 클릭하면 아래처럼 메뉴가 바뀌는데요, 여기서 Credentials를 클릭해주세요.



System의 Domain을 클릭해서, Add Credentials를 해주세요.

User: Minjeong Kim » Credentials

Dashboard » Minjeong Kim » Credentials

### Credentials

T	P Store	Domain	ID	Name
Stores scoped to User: Minjeong Kim				
	P Store	Domains		
	User: Minjeong Kim	(global)		
Stores from parent				
	P Store	Domains		
	System	(global)		

아이콘: S M L

REST API Jenkins 2.361.4

System » Global credentials (unrestricted)

Dashboard » Jenkins 관리 » Credentials » System » Global credentials (unrestricted) »

### Global credentials (unrestricted)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
This credential domain is empty. How about <a href="#">adding some credentials?</a>			

아이콘: S M L

REST API Jenkins 2.361.4

이때, 아래와 같이 채워주세요.



username: docker hub 아이디

p/w: docker hub 비밀번호

ID: docker-hub

Description은 단순히 설명란이기 때문에 안적어도 됩니다.



## New credentials

Kind

Username with password

Scope ?

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

Username ?

ensiqle

☐ Treat username as secret ?

Password ?

\*\*\*\*\*

ID ?

docker-hub

Description ?

docker-hub



Create

아래처럼 생성되어야 합니다.

### Global credentials (unrestricted)

+ Add Credentials

Credentials that should be available irrespective of domain specification to requirements matching.

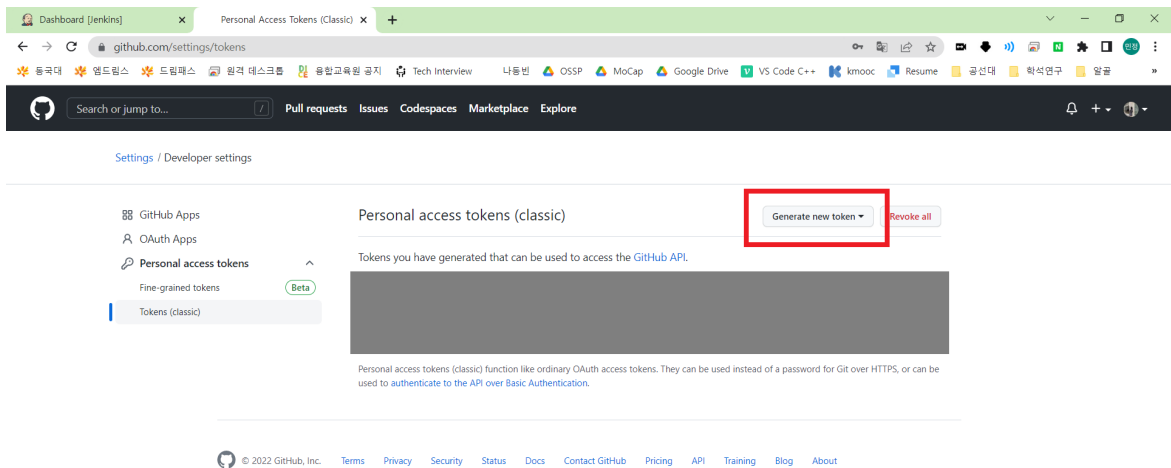
ID	Name	Kind	Description
 docker-hub	ensiqle/***** (docker-hub)	Username with password	docker-hub 

## Github-Jenkins 연동

### 6. Github과 Jenkins를 연동해주세요.

Github의 프로필 클릭 - Settings - Developer settings - Personal access token - Tokens(classic)으로 들어가셔서 token generate를 해주세요.

똑같이 classic token 생성해주시면 됩니다.



Note: Jenkins(본인이 알아볼 수 있는 이름 적어주세요.)  
Expiration: Token 소멸 날짜로, 저는 기본 30일 선택했습니다.

저는 아래와 같은 권한 선택하여서 token을 생성하였습니다. 권한을 더 주셔도 상관없지만, 최소한 이 3개는 선택해주셔야 합니다.

## 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 checked="" type="checkbox"/> admin:org	Full control of orgs and teams, read and write org projects
<input checked="" type="checkbox"/> write:org	Read and write org and team membership, read and write org projects
<input checked="" type="checkbox"/> read:org	Read org and team membership, read org projects
<input checked="" type="checkbox"/> manage_runners:org	Manage org runners and runner groups
<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

빨간 박스 처둔 곳에 본인의 token이 나옵니다. 해당 내용은 다시 볼 수 없으니 꼭 어딘가에 옮겨두세요!

만약 다시 보고싶으시다면, 새로 생성하셔야합니다.




## Personal access tokens (classic)

Generate new token ▼

Revoke all


Tokens you have generated that can be used to access the [GitHub API](#).

Make sure to copy your personal access token now. You won't be able to see it again!



WSL2 — admin:enterprise, admin:gpg\_key, admin:org, admin:org\_hook, admin:public\_key, admin:repo\_hook, delete:packages, delete\_repo, gist, notifications, repo, user, workflow, write:discussion, write:packages

Last used within the last 7 months

 This token has no expiration date.

Delete

Delete

ghp\_PZEVsgUjfh0E6SEBFnp7aNRH6Fsb2UaZF6

또 다시 credential을 생성해주시는데, 이번에는 Kind를 Secret text로 설정하시고 아까 복사한 Token Num을 secret에 넣어주시고, id는 알아서 설정해주세요.

## New credentials

Kind

Secret text

Scope ?

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

Secret

.....

ID ?

github

Description ?

github

Create

그러면 아래처럼 2개의 계정이 생성되었을 겁니다.

**Global credentials (unrestricted)** [+ Add Credentials](#)

Credentials that should be available irrespective of domain specification to requirements matching.

ID	Name	Kind	Description
docker-hub	ensiqle/***** (docker-hub)	Username with password	docker-hub
github	github	Secret text	github

## Jenkins Pipeline 만들기


7. 메인 페이지에서 Create Job을 해주세요.

이때, pipeline을 선택하여 제작해주세요.

### Enter an item name


hw6

» Required field



#### Freestyle project

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.



#### Pipeline

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.

아래와 같이 두 개의 옵션을 체크해주세요.

- Do not allow concurrent builds: build가 동시에 되지 않게 하는 옵션
- Github project: 빌드를 자동화할 프로젝트 링크

☒ Do not allow concurrent builds
 

☐ Abort previous builds
 ?

☐ Do not allow the pipeline to resume if the controller restarts

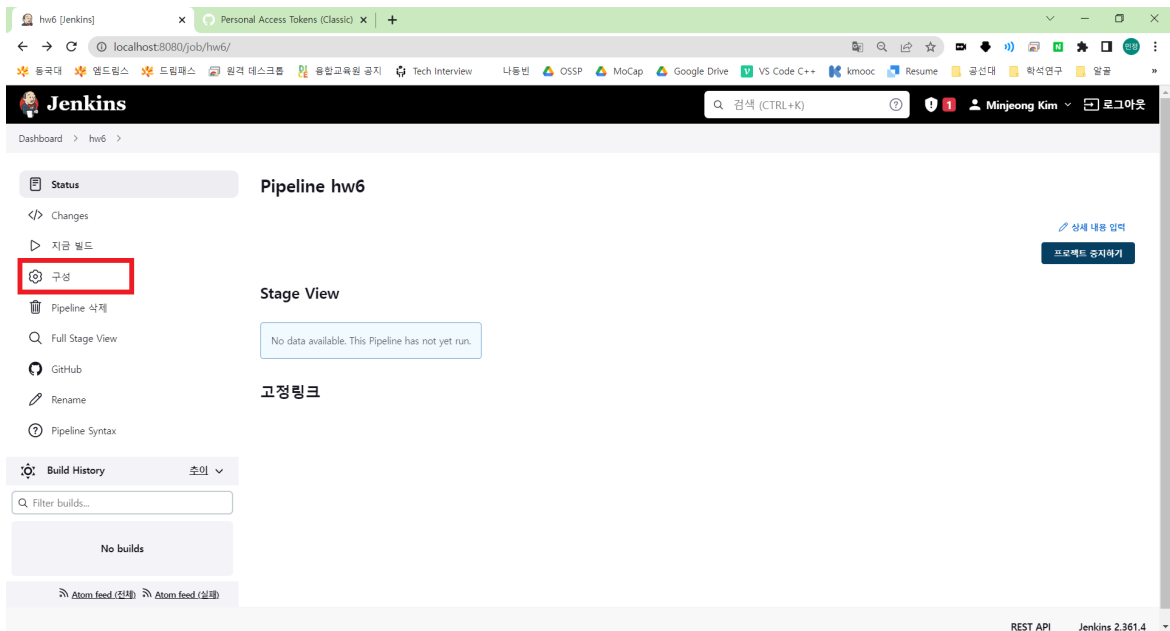
☒ GitHub project
 

Project url
 ?

https://github.com/kimminje0ng/oss\_p\_hw6

고급...

pipeline 생성 후 script를 작성해야합니다. 아래의 메뉴를 클릭한 뒤, Advanced Project Options - Pipeline에서 script를 작성하는 란을 찾아주세요.



## Pipeline

Definition

Pipeline script

Script ?

1

try sample Pipeline...

☒ Use Groovy Sandbox ?

[Pipeline Syntax](#)

작성하는 내용은 아래 8번 문항에서 설명할 예정입니다.

### 8. 아래 내용과 Docker compose 파일을 참고하여 스크립트를 완성해주세요.

Scripts는 아래의 6단계의 Stage로 구성됩니다. withCredentials의 경우 위에서 생성한 credentials와 본 파이프라인을 연결하기 위해 필요합니다.

1. **Pull:** git 소스를 다운로드합니다. 위에서 만든 자신의 프로젝트 git url을 넣어줍니다.
2. **Unit Test:** 본 프로젝트에서는 사용하지 않습니다.
3. **Build:** docker-compose build를 진행합니다.
4. **Tag:** docker image tag를 붙입니다.
5. **Push:** docker hub에 push합니다.
6. **Deploy:** docker-compose 명령어로 이미지를 실행합니다.

아래는 샘플 코드입니다. <<>>안에 적힌 내용을 수정하여 완성해주세요.

```

node{
  git pull: true, url:'<<github 링크>>'
  withCredentials([[${class: 'UsernamePasswordMultiBinding',
    credentialsId: 'docker-hub',
    usernameVariable: 'DOCKER_USER_ID',
    passwordVariable: 'DOCKER_USER_PASSWORD'}]]){
    stage('Pull'){
      git '<<github 링크>>'
    }

    stage('Unit Test'){
    }

    stage('Build'){
      sh(script: 'docker-compose build web')
    }

    stage('Tag'){
      sh(script: '''docker tag ${DOCKER_USER_ID}/<<dockerhub 레포이름>> ${DOCKER_USER_ID}/<<dockerhub 레포이름>>:${BUILD_NUMBER}''')
    }
    stage('Push'){
      sh(script: 'docker login -u ${DOCKER_USER_ID} -p ${DOCKER_USER_PASSWORD}')
      sh(script: 'docker push ${DOCKER_USER_ID}/<<dockerhub 레포이름>>:${BUILD_NUMBER}')
      sh(script: 'docker push ${DOCKER_USER_ID}/<<dockerhub 레포이름>>:latest')
    }
    stage('Deploy'){
      sh(script: 'docker-compose up -d production')
    }
  }
}

```

## 9. 빌드하기

pipeline 페이지에서 지금 빌드를 클릭하여 빌드가 제대로 되는지 확인해주세요. 빌드를 완료한 뒤 나오는 stage-view를 캡처하여 제출해주세요.

이때, 파이프라인 실행 전 저번 과제에서 업로드하셨던 yml 파일을 수정하고, github에 업로드 해주신 뒤 다음 과정 진행해주셔야 정상적으로 동작합니다!



### 어떤 것을 수정해야 할지에 대한 힌트

위 script를 보시면 `sh(script: 'docker login -u ${DOCKER_USER_ID} -p ${DOCKER_USER_PASSWORD}')` 를 사용해서 USER ID를 불러오는 것을 확인할 수 있습니다.

원래의 yml 코드는