

# 챕터 7 쿠버네티스 CI/CD 환경 구성과 활용

Created

2021년 12월 27일 오후 9:58

Tags

비어 있음

[CI/CD 구성을 위한 인스턴스를 새로 구성](#)

[계정 정보](#)

[클러스터 초기화](#)

[도커 및 도커 컴포즈 설치](#)

[harbor 컨테이너 레지스트리 설치와 활용](#)

[docker CLI를 활용한 Harbor 사용](#)

[gogs 설치 \(경량화된 gitlab\) 와 활용](#)

[Jenkins 설치](#)

[깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정](#)

[Jenkins에서 파이프라인 구성](#)

[Jenkins에 harbor registry 인증 정보 입력](#)

[빌드와 푸시를 위한 Jenkinsfile 설정 변경](#)

[Github 웹훅 테스트](#)

[Argo를 활용한 CD 구성](#)

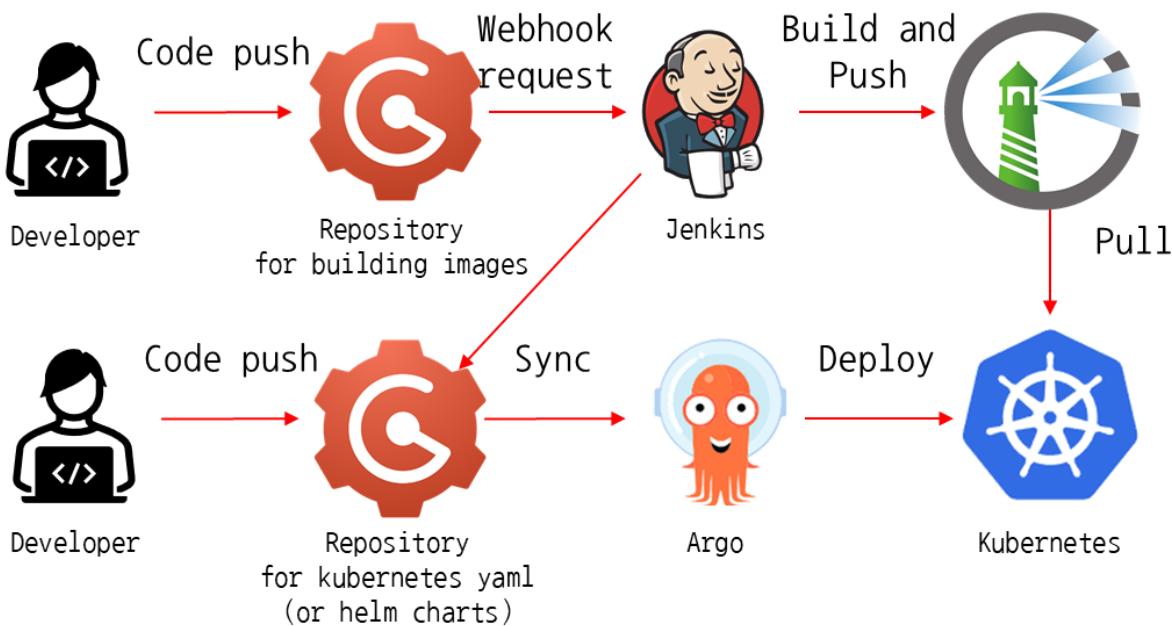
[실습에 필요한 git 레지스트리 마이그레이션](#)

[Argo설치](#)

[yaml 매니페스트 파일을 활용한 배포](#)

[헬름차트를 활용한 배포](#)

## CI/CD 구성을 위한 인스턴스를 새로 구성



## 계정 정보

- jenkins
  - id: jenkins
  - pw: test1234
- Harbor
  - id: admin
  - pw: Test1234
- Argo
  - id: admin
  - pw: <secret value>
- Gogs
  - id: gogs
  - pw: test1234

## 클러스터 초기화

- 1 마스터 노드 + 2 워커 노드
- 남은 하나를 CI/CD를 위한 Ubuntu 인스턴스로 구성 (4CPU//8GB 메모리)

## 도커 및 도커 컴포즈 설치

```
# 관리자 권한 sudo -i # docker 설치 apt update && apt install -y docker.io # 도커  
컴포즈 설치 sudo curl -L "https://github.com/docker/compose/releases/download/1.2  
9.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
docker rm --force `docker ps -a -q` docker volume rm --force `docker volume ls  
-q`
```

## harbor 컨테이너 레지스트리 설치와 활용

스크립트를 사용해서 설치 진행(harbor io)

```
wget https://gist.githubusercontent.com/kacole2/95e83ac84fec950b1a70b0853d6594  
dc/raw/ad6d65d66134b3f40900fa30f5a884879c5ca5f9/harbor.sh bash harbor.sh
```

우리는 도메인이 없으므로 IP를 선택한다.

```
1) IP 2) FQDN Would you like to install Harbor based on IP or FQDN? > 1번 선택
```

HTTPS 통신을 위한 인증서를 구성하고 설정하여 설치를 진행해야 한다. 다음 스크립트를 구성하고 실행해 CA와 Harbor에서 사용할 Certificate를 생성한다.

```
cd ~ mkdir pki cd pki # ca 키와 인증서 생성 openssl req -x509 -nodes -days 3650 -n  
ewkey rsa:2048 \ -out ca.crt \ -keyout ca.key \ -subj "/CN=ca" # harbor server  
키와 인증서 생성 openssl genrsa -out server.key 2048 openssl req -new -key server.  
key -out server.csr -subj "/CN=harbor-server" openssl x509 -req -in server.csr  
-CA ca.crt \ -CAkey ca.key \ -CAcreateserial -out server.crt -days 365 # 키와  
인증서 복제 mkdir -p /etc/docker/certs.d/server cp server.crt /etc/docker/certs.d/  
server/ cp server.key /etc/docker/certs.d/server/ cp ca.crt /etc/docker/certs.  
.d/server/ cp ca.crt /usr/local/share/ca-certificates/harbor-ca.crt cp server.  
crt /usr/local/share/ca-certificates/harbor-server.crt update-ca-certificates
```

harbor.yml 템플릿을 사용해 harbor의 설정을 구성한다.

```
cd ~/harbor cp harbor.yml.tpl harbor.yml vim harbor.yml
```

harbor.yml 파일의 내부에서 호스트 이름과 키, 인증서의 경로만 바꿔주면 된다. 여기서 초기 패스워드와 유저 아이디도 확인이 가능하다. 호스트 이름의 IP는 현재 인스턴스의 IP를 입력한다.

```
# 현재 인스턴스의 IP hostname: 10.0.2.7 # http related config http: # port for http, default is 80. If https enabled, this port will redirect to https port port : 80 # https related config https: # https port for harbor, default is 443 port: 443 # The path of cert and key files for nginx certificate: /etc/docker/certs.d/server.crt private_key: /etc/docker/certs.d/server/server.key
```

끝으로 준비 및 설치 스크립트를 실행한다. 준비 스크립트는 이미지를 준비하고 인증서 파일을 위한 설정을 구성한다. install.sh 파일은 도커 컴포즈를 사용해 harbor 실행에 필요한 컨테이너들을 배포한다.

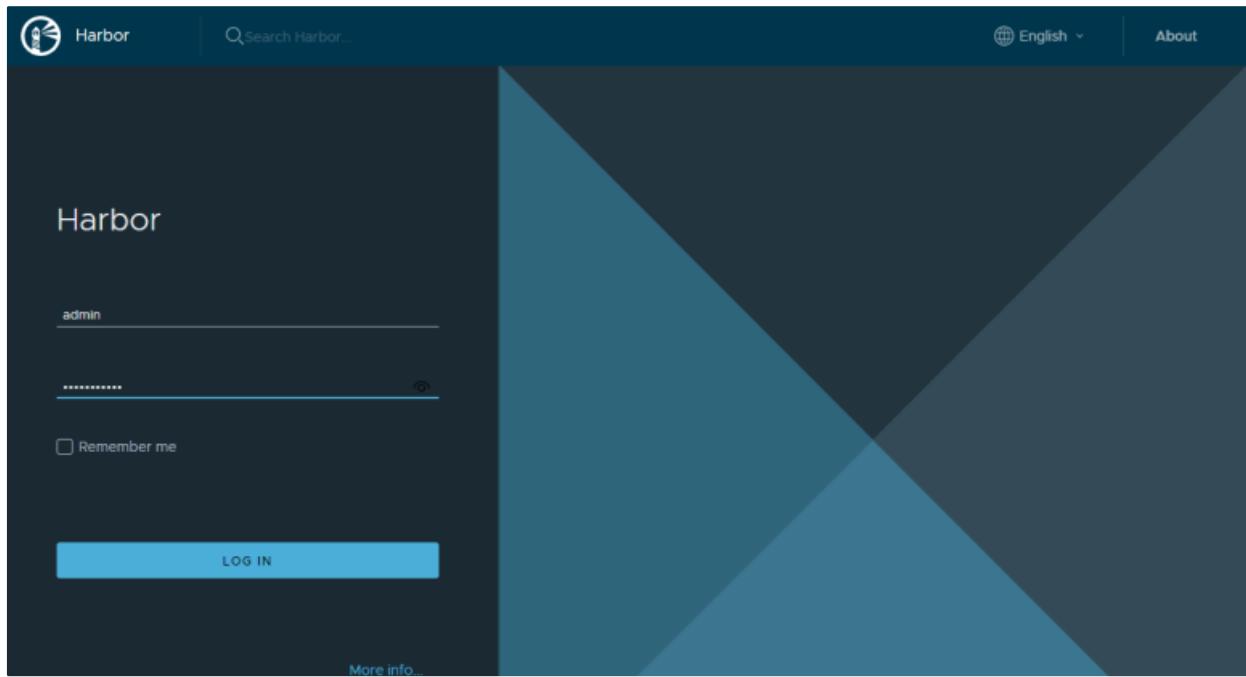
```
./prepare sudo curl -L "https://github.com/docker/compose/releases/download/1.29.2/docker-compose-$(uname -s)-$(uname -m)" -o /usr/local/bin/docker-compose  
./install.sh
```

끝으로 준비 및 설치 스크립트를 실행한다. 준비 스크립트는 이미지를 준비하고 인증서 파일을 위한 설정을 구성한다. install.sh 파일은 도커 컴포즈를 사용해 harbor 실행에 필요한 컨테이너들을 배포한다. 웹브라우저로 인스턴스 IP로 접속한다. 80포트는 443포트로 리다이렉션이 시킨다. 고급을 누르고 안전하지 않음을 선택한다.

The screenshot shows a browser window with the following details:

- Address bar: ▲ 안전하지 않음 | <https://34.134.229.160>
- Message: **비공개 연결이 아닙니다.** (This is not a public connection.)
- Description: 공격자가 34.134.229.160에서 사용자의 정보 (예: 암호, 메시지 또는 신용 카드)를 도용하려고 시도할 수 있습니다. (An attacker can try to steal your information such as password, message or credit card from 34.134.229.160.)
- Error code: NET::ERR\_CERT\_AUTHORITY\_INVALID
- Buttons: 고급 (Advanced) and 돌아가기 (Back)

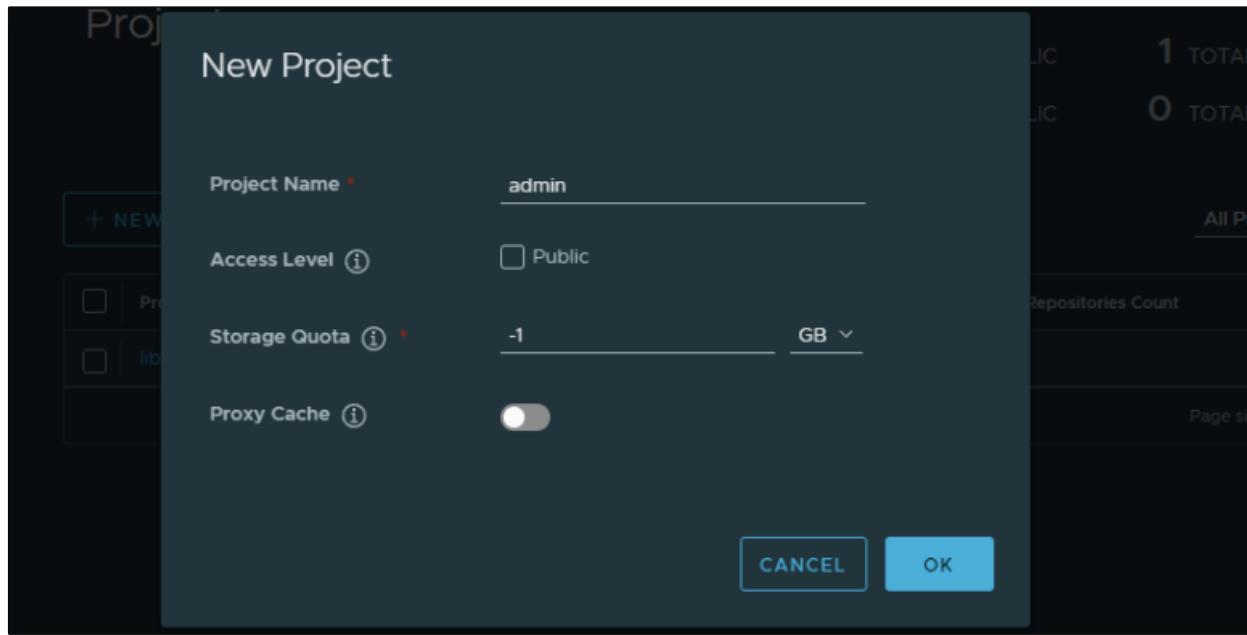
Harbor의 초기 ID와 패스워드는 admin//Harbor12345다. 접속을 수행한다.



로그인 후 패스워드를 변경할 수 있다. 처음 로그인하면 기본 프로젝트인 Library가 보인다. 이 프로젝트는 누구나 엑세스해서 사용할 수 있는 Public 모드로 구성되어 있다. 여기에 New Project 버튼을 눌러 새 프로젝트를 생성한다.

PROJECTS	0 PRIVATE	1 PUBLIC	1 TOTAL
REPOSITORIES	0 PRIVATE	0 PUBLIC	0 TOTAL

admin 프로젝트는 프라이비트로 구성한다.



프로젝트를 생성하고 admin 프로젝트로 가면 해당 기능을 사용할 수 있는 유저를 정할 수 있는 기능이 있다. 프로젝트 단위대로 권한을 부여해 액세스할 수 있는 유저를 구성할 수 있다. 유저는 왼쪽 유저탭에서 새로운 유저를 구성할 수 있다.

The screenshot shows the 'Members' tab for the 'admin' project. It lists one member: 'admin' with a 'User' member type and 'Project Admin' role. There are buttons for adding new users or groups.

## docker CLI를 활용한 Harbor 사용

이제 도커 CLI로 돌아온다. admin 유저를 사용해 원하는 이미지를 업로드해보자. 우선 로그인을 진행한다. 다음 명령을 실행하자.

```
docker login 127.0.0.1 -u admin -p Test1234
```

admin 권한으로 업로드를 진행한다. nginx를 pulling하고 nginx에 태그를 추가한다. 여기에 전달되는 IP는 Harbor의 IP다. 그리고 추가된 태그로 푸시를 진행한다.

```
docker pull nginx docker tag nginx 127.0.0.1/admin/nginx docker push 127.0.0.1/admin/nginx
```

푸시된 이미지는 웹에서도 확인이 가능하다.

The screenshot shows a user interface for managing repositories. At the top, there's a navigation bar with tabs: Summary, **Repositories** (which is currently active), Members, Labels, Scanner, P2P Preheat, Policy, Robot Accounts, Webhooks, Logs, and more. Below the navigation is a search bar with options for PUSH COMMAND, Q, and C. The main area displays a table of repositories. The columns are: Name, Artifacts, Pulls, and Last Modified Time. There is one entry: admin/nginx, which has 1 artifact and was last modified on 10/26/21, 1:28 AM. At the bottom right of the table, it says 'Page size 15 1 - 1 of 1 items'.

## gogs 설치 (경량화된 gitlab) 와 활용

도커 컴포즈를 활용해 gogs와 데이터베이스 설치

```
mkdir ~/gogs cd ~/gogs wget https://gist.githubusercontent.com/ahromis/4ce4a58623847ca82cb1b745c2f83c82/raw/31e8ced3d7e08c602a1c0ca8994c063994971c7f/docker-compose.yml vim docker-compose.yml
```

데이터베이스의 ID와 비밀번호 설정

```
version: '2' services: postgres: image: postgres:9.5 restart: always environment: - "POSTGRES_USER=gogs" - "POSTGRES_PASSWORD=test1234" - "POSTGRES_DB=gogs" volumes: - "db-data:/var/lib/postgresql/data" networks: - gogs gogs: image: gogs/gogs:latest restart: always ports: - "10022:22" - "3000:3000" links: - postgres environment: - "RUN_CROND=true" networks: - gogs volumes: - "gogs-data:/data" depends_on: - postgres networks: gogs: driver: bridge volumes: db-data: driver: local gogs-data: driver: local
```

```
docker-compose -f docker-compose.yml up -d
```

3000번 포트로 서비스를 실행 중인 것을 확인하고 접속할 수 있다.

```
docker ps -a
```

Gogs를 Docker에서 운영하고 있다면 [안내](#)를 읽고 변경해 주세요!

### 데이터베이스 설정

Gogs는 MySQL, PostgreSQL, SQLite3, MSSQL 또는 TiDB를 필요로 합니다.

데이터베이스 유형 *	PostgreSQL
호스트 *	postgres:5432
사용자 *	gogs
비밀번호 *	.....
데이터베이스 이름 *	gogs

MySQL에서는 utf8\_general\_ci 캐릭터셋으로 INNODB엔진을 이용해 주세요

### ▼ 관리자 계정 설정

ID가 1인, 첫번째로 생성된 계정이 관리자 계정이 되므로, 지금 계정을 생성하지 않으셔도 됩니다.

이름	gogs
비밀번호	.....
비밀번호 확인	.....
관리자 이메일	test@test.com

**Gogs 설치하기**



**새 저장소**

소유자 *	 gogs
저장소 이름 *	flask-example
좋은 저장소 이름은 짧고 기억하기 좋은 유니크한 키워드로 이루어 집니다.	
가시성	<input checked="" type="checkbox"/> 이 저장소는 <b>비공개</b> 저장소입니다 <input type="checkbox"/> This repository is <b>Unlisted</b>
설명	저장소 설명: 최대 512길이의 문자열이 가능합니다. 가능한 문자열입니다.: 512
.gitignore	.gitignore 서식을 선택합니다
라이센스	Apache License 2.0
Readme	Default
<input type="checkbox"/> 선택한 파일과 템플릿으로 이 저장소를 초기화 합니다.	
<b>저장소 만들기</b> <b>취소</b>	

## git 프로그램 설치 및 사용자 설정

```
apt update && apt install git -y git config --global user.name gogs git config --global user.email test@test.com
```

기존 github의 있는 자료를 가져와서 gogs에 업로드해보자.

```
git clone https://github.com/gasbugs/flask-example cd flask-example/ rm -rf .git/ git init git add . git commit -m "refresh commit" git remote add origin http://127.0.0.1:3000/gogs/flask-example.git git push -u origin master
```

## Jenkins 설치

빠르고 쉬운 설치를 위해 도커를 설치하고 도커 이미지를 사용해 Jenkins를 배포한다. Jenkins를 배포할 때는 일부 디렉토리를 공유하도록 설정했으며 도커 소켓 또한 공유하도록 구성한다. 이 소켓을 사용해 Jenkins는 호스트에 설치된 도커 기능을 사용할 수 있다.

```
# 도커를 사용해 jenkins 구성 및 도커 소켓 공유 docker run -d -p 8080:8080 --name jenkins -v /home/jenkins:/var/jenkins_home -v /var/run/docker.sock:/var/run/docker.sock -u root jenkins:jenkins:lts
```

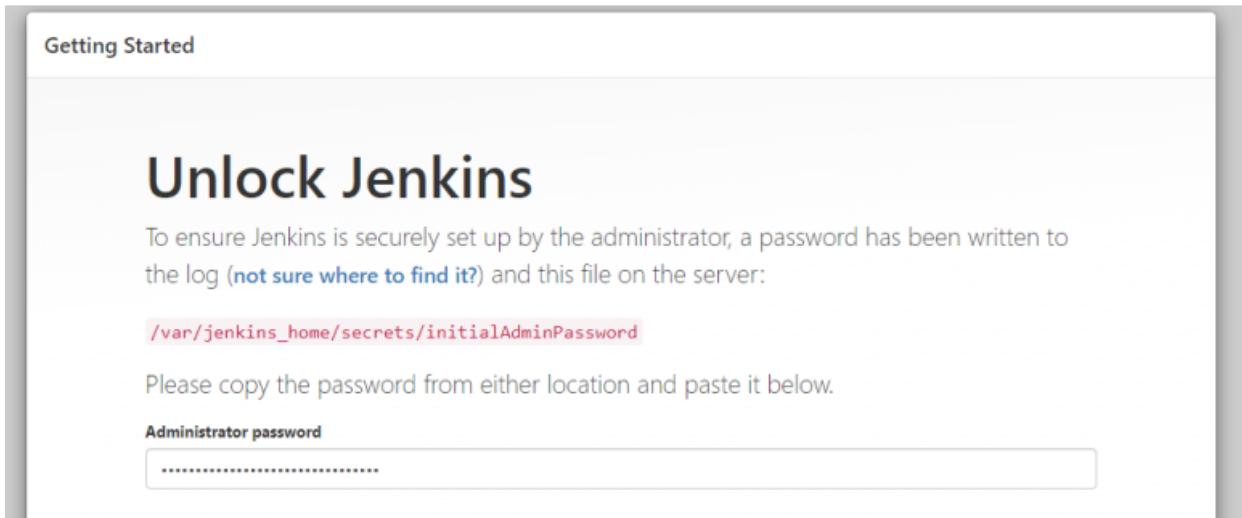
Jenkins에도 docker 클라이언트가 필요하므로 도커를 설치하자.

```
# jenkins에 docker client 설치 docker exec jenkins apt update docker exec jenkins apt install -y docker.io
```

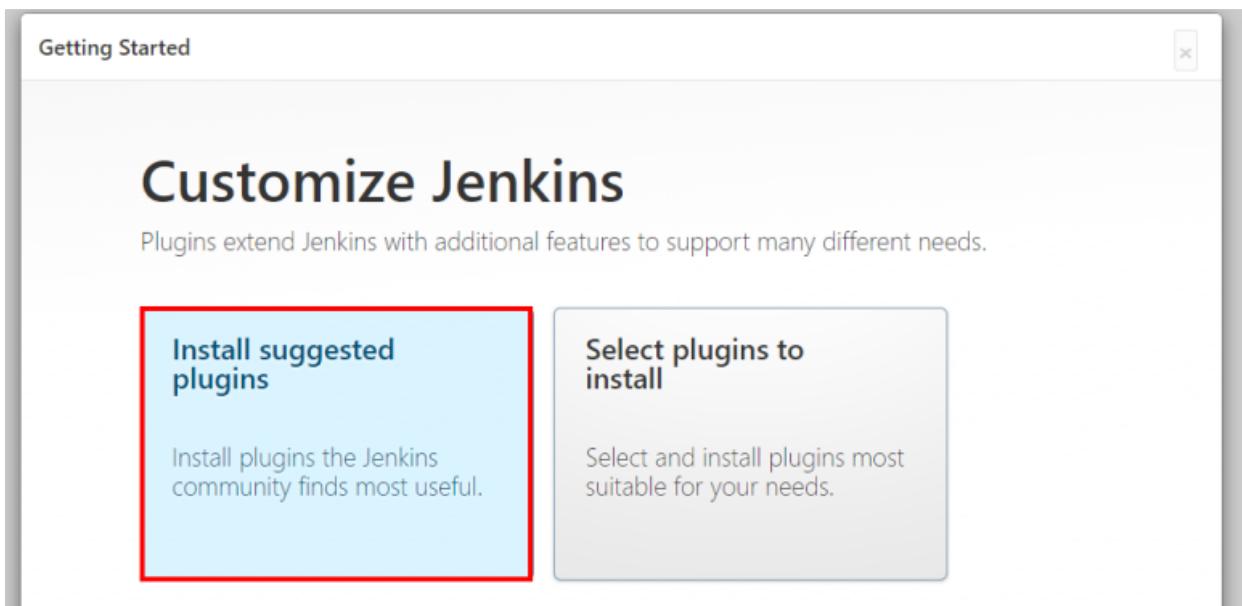
Jenkins의 초기화에 사용할 패스워드를 조회한다.

```
# 젠킨스 초기패스워드 조회 docker exec jenkins cat /var/jenkins_home/secrets/initialAdminPassword
```

Jenkins가 80포트로 서비스 중일 것이다. 앞서 확인한 인스턴스의 IP를 사용해 브라우저로 접속한다. 그리고 초기패스워드를 입력하자.



초기 설치는 Install suggested plugins를 사용한다.



플러그인 설치가 완료되면 유저를 설정할 수 있는 페이지가 나타난다. 적절한 유저로 구성하도록 하자.

Getting Started

## Create First Admin User

계정명: user0

암호: .....  
암호 확인: .....

이름: test

이메일 주소: test@test.com

인스턴스의 URL은 자동으로 입력된다. 서비스하고 있는 IP로 수정하고 진행한다.

Getting Started

## Instance Configuration

Jenkins URL: http://34.135.119.94/

The Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources. That means this value is required for proper operation of many Jenkins features including email notifications, PR status updates, and the BUILD\_URL environment variable provided to build steps.

The proposed default value shown is not saved yet and is generated from the current request, if possible. The best practice is to set this value to the URL that users are expected to use. This will avoid confusion when sharing or viewing links.

jenkins 구성이 완료되었다!

Jenkins

Dashboard >

새로운 Item

사람

빌드 기록

Jenkins 관리

My Views

Lockable Resources

New View

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중  
2 대기 중

Jenkins에 오신 것을 환영합니다.

This page is where your Jenkins jobs will be displayed. To get started, you can set up distributed builds or start building a software project.

Start building your software project

Create a job →

Set up a distributed build

Set up an agent →

Configure a cloud →

Learn more about distributed builds

도커 파이프라인, gogs webhook 플러인 설치

도커를 빌드하는데 사용하는 파이프라인 플러그인을 설치하자. 왼쪽 메뉴에서 Jenkins 관리를 클릭하고 플러그인 관리 메뉴로 들어간다.

The screenshot shows the Jenkins System Configuration page. On the left sidebar, 'Jenkins 관리' is selected. In the main content area, there's a yellow banner at the top with the text: 'Building on the controller node can be a security issue. You should set up distributed builds. See the documentation.' Below it, there are two sections: 'System 설정' (System Configuration) and '노드 관리' (Node Management). The 'System 설정' section has a sub-section titled 'Global Tool Configuration' with the description: 'Configure tools, their locations and automatic installers.' To the right, there's a box titled '플러그인 관리' (Plugin Management) with the sub-description: 'Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.' A red box highlights this 'Plugin Management' section.

The screenshot shows the Jenkins Plugin Manager page. The left sidebar shows 'Plugin Manager' is selected. The main area has a search bar with 'docker pipeline'. Below it, there are tabs: '업데이트된 플러그인 목록' (Updated Plugins), '설치 가능' (Installable), '설치된 플러그인 목록' (Installed Plugins), and '고급' (Advanced). A table lists a single plugin: 'Docker Pipeline' (version 1.26). The 'Install' button is highlighted with a red box. Below the table, there are buttons for 'Download now and install after restart' and '자금 확인' (Check Now). At the bottom, there's a 'Filter' input field with 'Gogs Plugin' typed in. The URL in the browser address bar is 192.168.106.100:8080/pluginManager/available.

## 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

gogs에서 변경사항이 있을 때마다 jenkins에 알릴 수 있도록 Webhook을 설정해보자. 레파지토리의 Settings - Webhooks에 1편에서 생성한 젠킨스 인스턴스 URL 주소와 /gogs-webhook을 함께 작성하고 저장한다. url 형식은 다음과 같다.

```
http://your-gogs-repo/your-username/your-project/settings/hooks
```

설정

옵션  
공동 작업  
브랜치  
Webhooks  
Git Hooks  
배포 키

Webhooks

웹후크는 기본적인 HTTP POST 이벤트 트리거입니다. Gogs에서 무슨 일이 발생할 때마다, 지정한 대상 호스트에 알림을 보냅니다. 웹후크 안내서에서 자세히 알아보십시오.

✓ http://10.0.2.7:8080/gogs-webhook/?job=flask-example-docker-pipeline

Add a new webhook: Choose a type... ▾

## Jenkins에서 파이프라인 구성

Jenkins에서 Webhook 이벤트를 받아 빌드를 시작할 수 있도록 구성해보자. 새로운 Item을 클릭한다.

Dashboard

새로운 Item

사람

빌드 기록

이름을 적절하게 구성하고 pipeline을 선택하자. 이름은 flask-example-docker-pipeline으로 정했다.

Enter an item name

flask-example-docker-pipeline

» 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.

General에는 github 프로젝트를 선택하고 프로젝트 URL 정보를 입력한다. gogs의 레파지토리를 입력해야 한다.

**General** Build Triggers Advanced Project Options Pipeline

설명

[Plain text] 미리보기

Do not allow concurrent builds  
 Do not allow the pipeline to resume if the controller restarts  
 GitHub project

Project url <https://github.com/gasbugs/flask-example> [고급...](#)

Pipeline speed/durability override  
 Preserve stashes from completed builds  
 Throttle builds  
 오래된 빌드 삭제  
 이 빌드는 매개변수가 있습니다

빌드 트리거는 Webhook을 통해서 시작할 수 있도록 설정하자.

### Build Triggers

Build after other projects are built  
 Build periodically  
 GitHub hook trigger for GITScm polling  
 Poll SCM  
 빌드 안함  
 Quiet period  
 빌드를 원격으로 유발 (예: 스크립트 사용)

파이프라인 정보로 사용할 flask-example의 데이터를 설정하도록 하자. Credentials는 Add 버튼을 눌러 새로 생성한다.

### Pipeline

Definition

Pipeline script from SCM

SCM

Git

Repositories

Repository URL <https://github.com/gasbugs/flask-example>

Credentials [Add](#)

gogs에서 ID//PW 발급 받아 다음과 같이 입력한다. 토큰 발급에 대한 상세 절차는 생략한다. (github에서는 토큰을 사용해야 함)

- username: gogs ID
- password: 패스워드
- gogs-cred: 젠킨스에서 사용할 Credential ID

Jenkins Credentials Provider: Jenkins

### Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope

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

Username

gasbugs

Treat username as secret

Password

.....

ID

github\_cred

Description

Add Cancel

그러면 다음과 같이 credentials 내용을 채울 수 있다.

Pipeline

Definition

Pipeline script from SCM

SCM

Git

Repositories

Repository URL

https://github.com/gasbugs/flask-example

Credentials

gasbugs/\*\*\*\*\*\*\*\* Add

이제 남은 부분을 구성해보자. main 정보를 입력하고 Jenkinsfile에 대한 위치 정보는 그대로 둔다. flask-example/Jenkinsfile의 파이프라인 정보를 읽어서 빌드 절차를 진행할 것이다. 모든 것이 완료되면 "저장"버튼을 누른다.

Branches to build

Branch Specifier (blank for 'any')

Add Branch

Repository browser

(자동)

Additional Behaviours

Add ▾

Script Path

Jenkinsfile

Lightweight checkout

Pipeline Syntax

## Jenkins에 harbor registry 인증 정보 입력

Dashboard ➔

빌드 기록

**Jenkins 관리**

- My Views
- Lockable Resources
- New View

빌드 대기 목록

빌드 대기 항목이 없습니다.

빌드 실행 상태

1 대기 중  
2 대기 중

building on the controller node can be a security issue. You should set up distributed builds. See [the documentation](#).

### System Configuration

시스템 설정

환경변수 및 경로 정보들을 설정합니다.

노드 관리

Add, remove, control and monitor the various nodes that Jenkins runs jobs on.

Global Tool Configuration

Configure tools, their locations and automatic installers.

### Security

Configure Global Security

Secure Jenkins; define who is allowed to access/use the system.

**Manage Credentials**

Configure credentials

Jenkins 스토어로 진입한 후에 Global credentials로 계속 접근한다.

## Stores scoped to Jenkins

P	Store ↓	Domains
	<b>Jenkins</b>	(global)

The screenshot shows the Jenkins System configuration page. At the top, there's a navigation bar with tabs for 'Domain' and 'Description'. Below this, a table lists a single item: 'Global credentials (unrestricted)' with a description: 'Credentials that should be available irrespective of domain specification to requirements matching.' There are size options 'S M L' at the bottom.

여기에서 앞서 생성한 `github_cred`가 보이는데 추가 인증 정보를 생성하도록 하자.

The screenshot shows the Jenkins Credentials page under 'Global credentials (unrestricted)'. On the left, there's a sidebar with 'Add Credentials' highlighted. The main area shows a table with one existing entry: 'github\_cred' (gasbugs/\*\*\*\*\*), 'Username with password'. There are size options 'S M L' at the bottom.

`username`과 `password`에 도커 허브의 ID와 패스워드를 차례로 입력하고 ID 부분에는 `docker-hub`라고 채워넣는다. ID는 반드시 `docker-hub`로 입력해야 한다. `Jenkinsfile`에서 이 ID를 참조하도록 구성되어 있다.

The screenshot shows the 'Add Credentials' form for a 'Username with password' type. It includes fields for 'Scope' (Global), 'Username' (gasbugs), and 'Password' (redacted). A checkbox 'Treat username as secret' is unchecked. The 'ID' field is set to 'docker-hub', which is highlighted with a red box. There are help icons (?) next to each field.

## 빌드와 푸시를 위한 Jenkinsfile 설정 변경

이제 코드를 변경해서 Github의 웹훅이 적절히 전달되는지 확인해보자. 여기서는 github 사이트에서 바로 수정해서 적용하도록 할 예정이다. `flask-example/Jenkins` 파일을 선택하고 수정 버튼을 누른다.

flask-example / Jenkinsfile

o4a test

Latest commit 642c104 yesterday History

1 contributor

26 lines (24 sloc) | 599 Bytes

```
1 node {
2     stage('Clone repository') {
3         checkout scm
4     }
5     stage('Build image') {
6         app = docker.build("gasbugs/flask-example")
7     }
8     stage('Push image') {
9         docker.withRegistry('https://registry.hub.docker.com', 'docker-hub') {
10            app.push("${env.BUILD_NUMBER}")
11            app.push("latest")
12        }
13    }
14 }
15
16 stage('Build image') {
17     app = docker.build("gasbugs/flask-example")
18 }
19
20 stage('Push image') {
21     docker.withRegistry('https://registry.hub.docker.com', 'docker-hub')
22     {
23         app.push("${env.BUILD_NUMBER}")
24         app.push("latest")
25     }
26 }
```

필자의 기존 docker hub 주소와 아이디를 harbor 주소와 ID로 변경하기 바란다. 수정이 완료되면 하단에 commit changes 버튼을 클릭한다.

```
node { stage('Clone repository') { checkout scm } stage('Build image') { app = docker.build("10.0.2.7/admin/flask-example") } stage('Push image') { docker.withRegistry('https://10.0.2.7', 'harbor-cred') { app.push("${env.BUILD_NUMBER}") app.push("latest") } } stage('Build image') { app = docker.build("10.0.2.7/admin/flask-example") } stage('Push image') { docker.withRegistry('https://10.0.2.7', 'harbor-cred') { app.push("${env.BUILD_NUMBER}") app.push("latest") } }
```

코드를 바꿨지만 젠킨스로 가보면 빌드가 진행되지 않고 있다. 첫 빌드는 직접 진행해주어야 한다. Build now 버튼을 눌러서 정상적으로 실행되는지 확인한다. 아래와 같이 모든 내용이 초록색으로 구성되면 정확하게 도커 레파지토리를 가져와서 도커 이미지를 만들고 푸시까지 수행한 것이다.

Clone repository	Build image	Push image	Build image	Push image
904ms	21s	7s	0ms	0ms
Oct 23 11:53	No Changes			
904ms	42s	9s	390ms	4s

고정링크  
Last build, (#1), 26 sec 전

## Github 웹훅 테스트

첫 빌드가 이뤄진 뒤에는 푸시된 정보를 실시간으로 받아서 빌드를 진행한다. 이제 file을 삭제하거나 생성해서 webhook 이벤트를 보내보자. 그럼 자동으로 웹훅으로 전달 받은 데이터로 인해 전체 빌드 과정을 다시 한번 진행하게 된다.

Clone repository	Build image	Push image	Build image	Push image
708ms	14s	7s	0ms	0ms
Oct 23 12:05	1 commit			
512ms				
Oct 23 11:53	No Changes			
904ms	42s	9s	390ms	4s

고정링크  
Last build, (#1), 26 sec 전

## Argo를 활용한 CD 구성

실습에 필요한 git 레지스트리 마이그레이션

<https://github.com/gasbugs/flask-example-apps> <https://github.com/gasbugs/helm-charts>

## Argo설치

argo를 쿠버네티스에 배포하자. 쿠버네티스에 argo를 배포하면 argo는 설치된 kube-api와 통신하며 클러스터를 관리한다. yaml 파일 내에 필요한 권한이 설정되어 있다.

```
kubectl create namespace argocd
kubectl apply -n argocd -f https://raw.githubusercontent.com/argoproj/argo-cd/stable/manifests/install.yaml
```

argocd에 잘 배포됐는지 확인한다.

```
$ kubectl get svc,pod -n argocd
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/argocd-dex-server   ClusterIP   10.8.10.139 <none>
5556/TCP,5557/TCP,5558/TCP 35m
service/argocd-metrics   ClusterIP   10.8.7.230
<none> 35m
service/argocd-redis     ClusterIP   10.8.12.102 <none> 6379/TCP
35m
service/argocd-repo-server ClusterIP   10.8.2.139 <none> 8081/TCP,8084/TCP
34m
service/argocd-server    ClusterIP   10.8.9.12 <none> 80/TCP,443/TCP 34m
service/argocd-server-metrics ClusterIP   10.8.13.88 <none> 8083/TCP 34m
NAME          READY   STATUS   RESTARTS   AGE
pod/argocd-application-controller-0 1/1   Running 0 34m
pod/argocd-dex-server-6c55787bc6-hrj4v 1/1   Running 0 34m
pod/argocd-redis-74d8c6db65-wzqfq 1/1   Running 0 34m
pod/argocd-repo-server-6c44847cf9-b9n6l 1/1   Running 0 34m
pod/argocd-server-67b65559fb-scgmp 1/1   Running 0 34m
```

로드밸런서로 변경하고 서비스의 IP를 확인한다. 변경되는데는 1분 정도 소요된다.

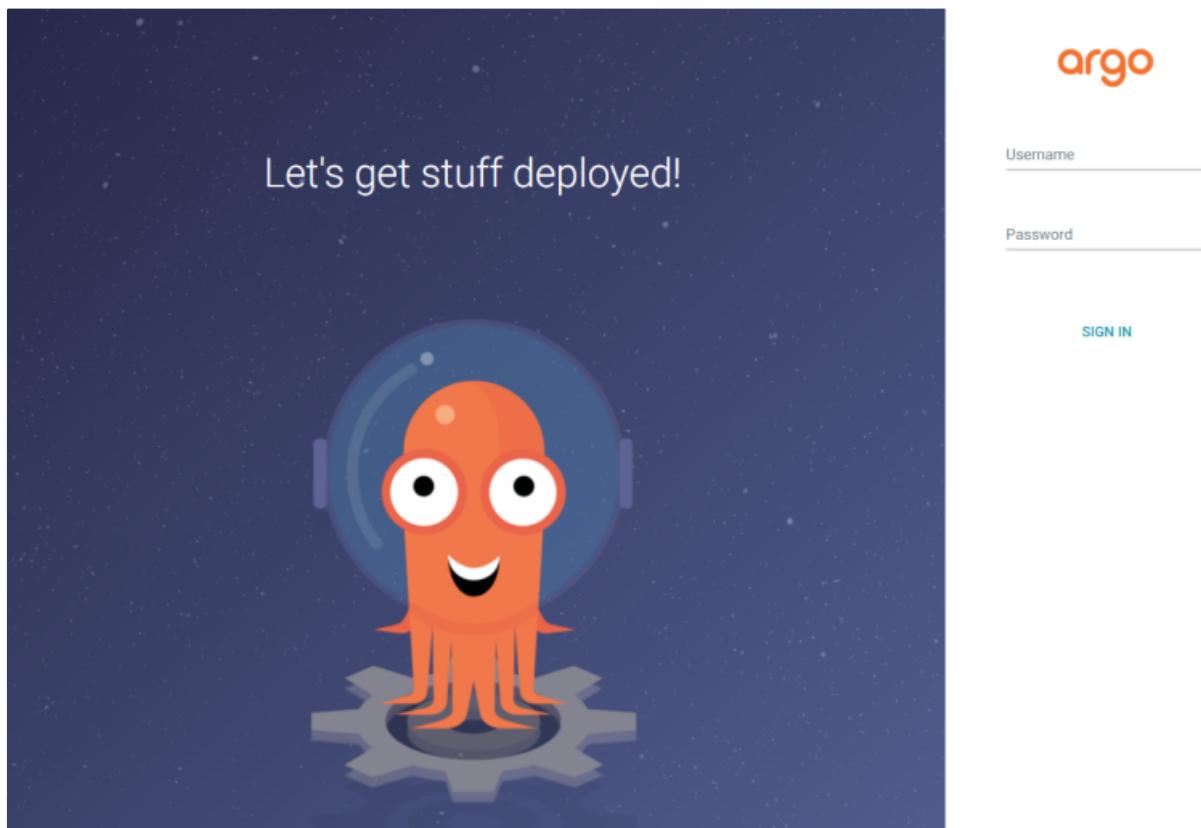
```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "NodePort"}}'  
kubectl get svc argocd-server -n argocd -w // 출력 NAME TYPE CLUSTER-IP  
EXTERNAL-IP PORT(S) AGE argocd-server NodePort 10.8.9.12 <pending>  
80:32524/TCP,443:31837/TCP 35m
```

계정의 ID는 admin이다. 패스워드는 시크릿으로 구성되어 있다.

```
kubectl -n argocd get secret argocd-initial-admin-secret -o jsonpath=".data.password" | base64 -d
```

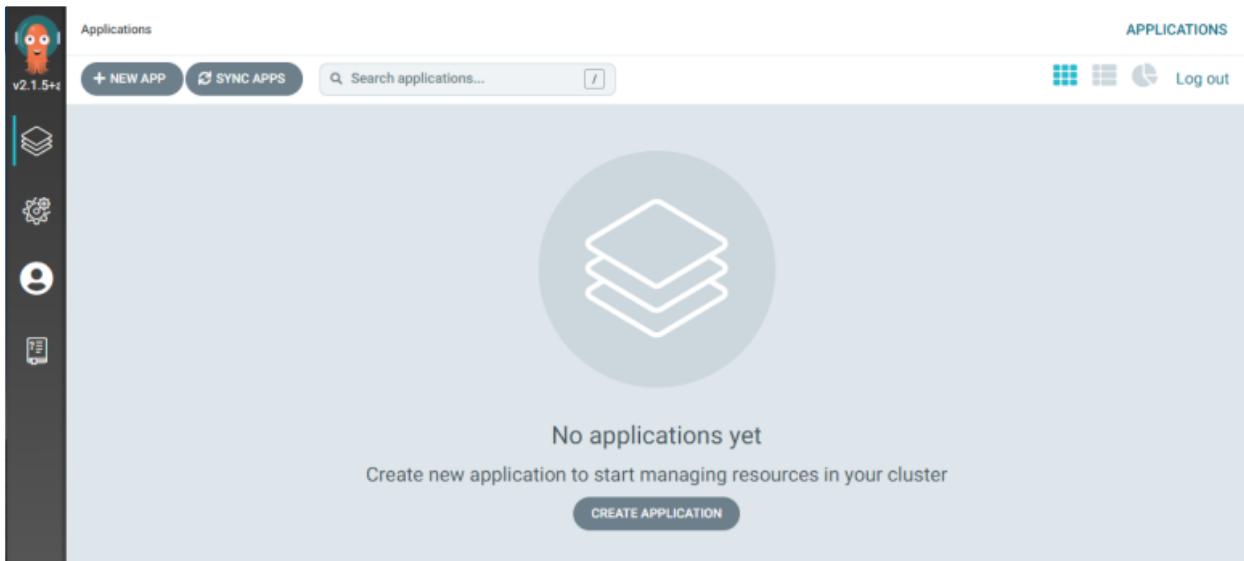
마스터노드의 IP의 31837로 접속한다. 그리고 앞서 확인한 패스워드를 입력해 로그인을 수행한다.

- username: admin
- password: <secret을 통해 확인한 값>



## yaml 매니페스트 파일을 활용한 배포

여기서 New APP 버튼을 누르고 새로운 앱을 추가하자.



애플리케이션 이름은 flask-example로 결정하고 Project는 default로 지정한다. 싱크 방식은 수동으로 할 수 있도록 Manual을 선택한다. 자동 동기화는 Git에 설정된 매니페스트 내용과 현재 애플리케이션의 상태가 다르거나 정상동작하지 않으면 자동으로 동기화해 애플리케이션을 다시 배포하거나 재구성하는 기능이다. 여기서는 수동으로 배포를 진행해본다. 끝으로 자동으로 네임스페이스를 생성할 수 있도록 옵션을 선택한다.

A screenshot of the 'flask-example' application configuration page. It shows the 'GENERAL' tab with the application name set to 'flask-example' and the project set to 'default'. The 'SYNC POLICY' section is set to 'Manual'. In the 'SYNC OPTIONS' section, the 'SKIP SCHEMA VALIDATION' and 'PRUNE LAST' checkboxes are unselected, while 'AUTO-CREATE NAMESPACE' and 'APPLY OUT OF SYNC ONLY' are selected. The 'PRUNE PROPAGATION POLICY' dropdown is set to 'foreground'. The 'REPLACE' checkbox is also unselected.

매니페스트 파일의 소스의 위치를 지정한다. flask-example-apps 프로젝트 링크를 전달하고 main 브랜치를 사용하도록 설정하자. Path는 배포할 yaml 파일이 있는 디렉토리를 지정한다. 우리의 프로젝트에는 flask-example-deploy 아래에 필요한 yaml이 구성되어 있다.

SOURCE

Repository URL  
<https://github.com/gasbugs/flask-example-apps> GIT ▾

Revision  
main Branches ▾

Path  
flask-example-deploy

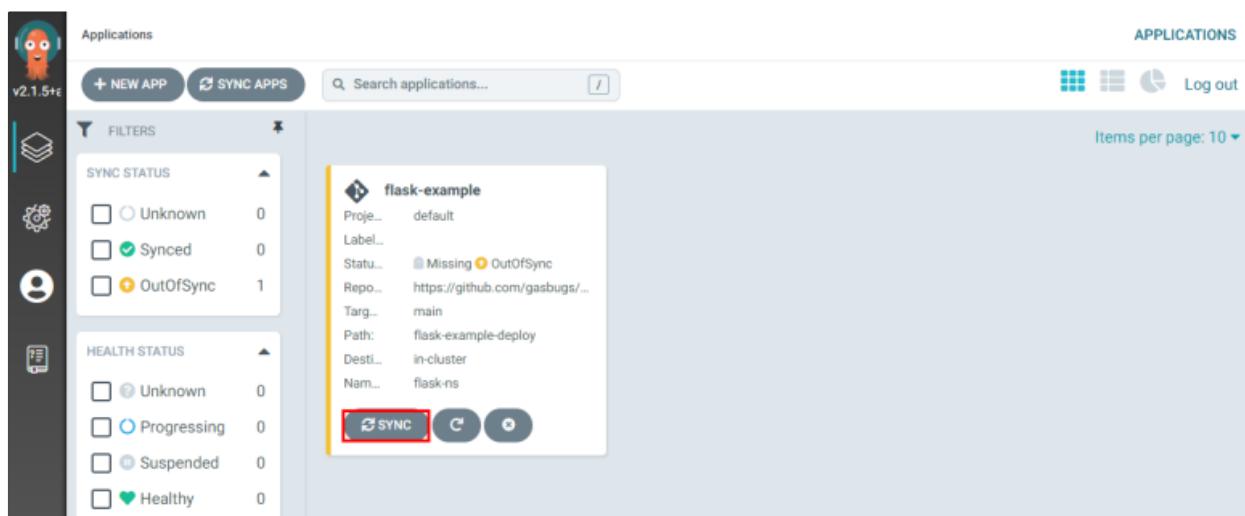
쿠버네티스 API 서버에 대한 정보를 입력한다. 우리가 구성한 argo는 쿠버네티스 내에 구성되었으므로 kube-apiserver에 대한 URL 정보를 도메인 주소로 입력할 수 있다. 네임스페이스는 애플리케이션을 배포할 공간을 의미한다. jenkins-ns에 배포를 진행하기 위해 다음과 같이 빈칸을 채운다.

DESTINATION

Cluster URL  
<https://kubernetes.default.svc> URL ▾

Namespace  
flask-ns

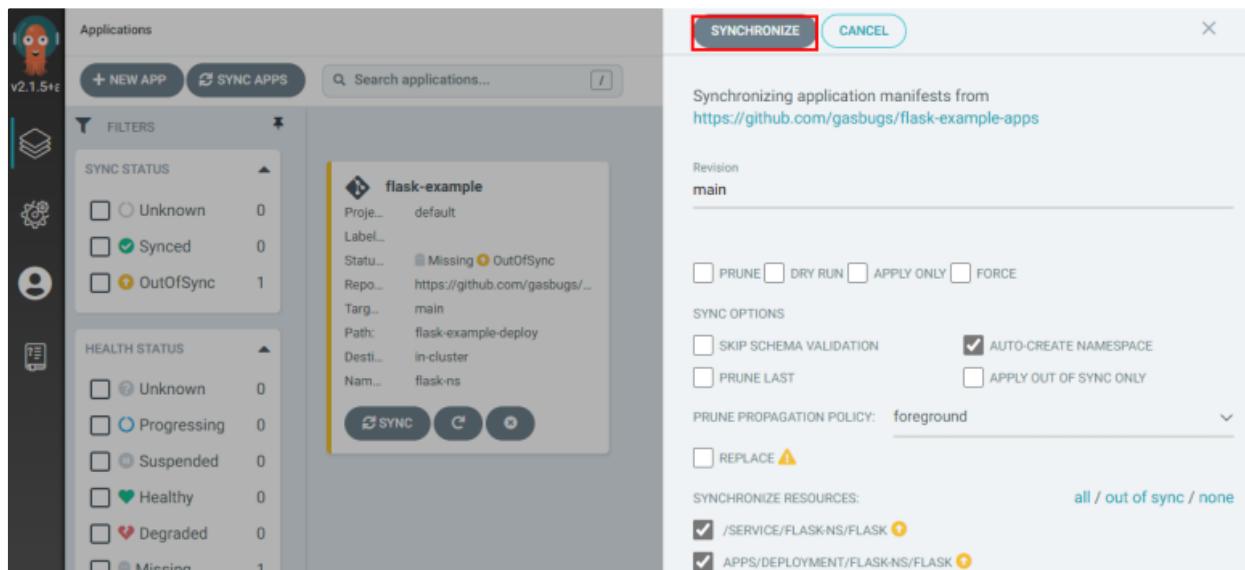
모든 설정이 완료되면 상단에 CREATE 버튼을 누른다. 새로 생성된 app인 flask-example을 확인할 수 있다. SYNC 버튼을 눌러서 배포를 진행해보자.



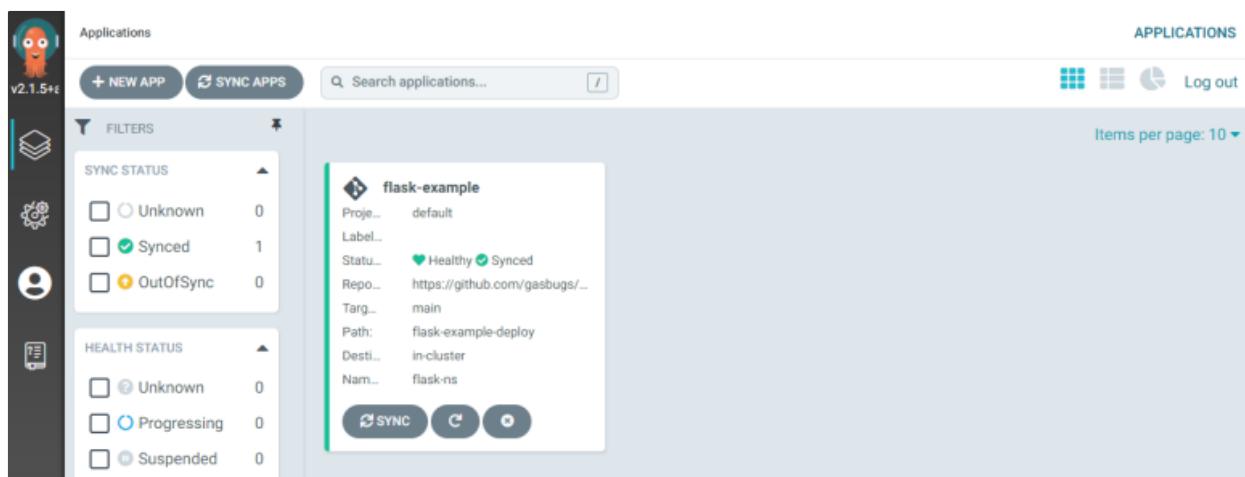
**flask-example**

- Proj... default
- Label...
- Status... Missing OutOfSync
- Repo... [https://github.com/gasbugs/...](https://github.com/gasbugs/)
- Targ... main
- Path: flask-example-deploy
- Desti... in-cluster
- Nam... flask-ns

**SYNC**



시간이 조금 지나면 헬스체크와 싱크 상태를 알려주는 모습으로 변경된다.

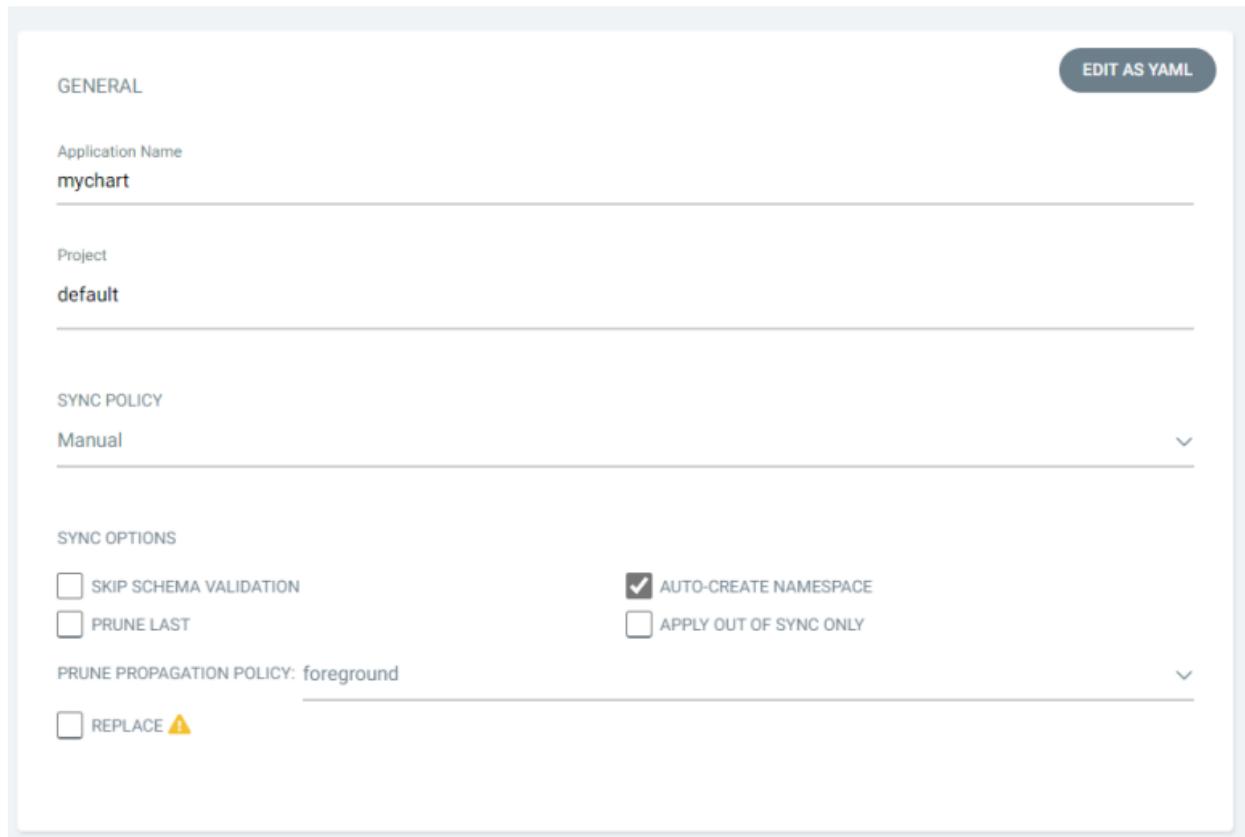


애플리케이션이 정상적으로 배포됐는지 다음 명령을 사용해 확인해보자.

```
$ kubectl get svc,deploy -n flask-ns
NAME           TYPE        CLUSTER-IP      EXTERNAL-IP   PORT(S)          AGE
service/flask  ClusterIP  10.8.10.189 <none>       80/TCP        2m6s
deployment.apps/flask  1/1  1  1  2m6s
```

## 헬름차트를 활용한 배포

이번에는 헬름차트를 활용해 배포를 진행해보자. 앞서 구성한 방법과 동일하게 애플리케이션 이름과 몇 정보를 입력한다.



그리고 앞에서 소개한 helm-charts 레파지토리에서 index.yaml 파일을 찾자. 경로는 stable/index.yaml이다. 이 파일을 Raw 데이터로 가져오기 위해 마우스 우클릭 후 링크를 복사한다.

The screenshot shows the GitHub repository for 'helm-charts' with a specific commit. The 'stable/index.yaml' file is displayed. The file content is as follows:

```

1 apiVersion: v1
2 entries:
3   mychart:
4     - apiVersion: v2
5     appVersion: 1.16.0
6     created: "2021-09-24T08:23:34.45503692Z"
7     description: A Helm chart for Kubernetes
8     digest: cc2b781cb642ee65a4084f85865ac42c87f0d55905999b92d42ce42ba24df02b
9     name: mychart
10    type: application
11    urls:
12      - mychart-0.1.0.tgz
13    version: 0.1.0
14    mychart2:
15      - apiVersion: v2

```

To the right of the file content, there is a context menu with various options. The 'Raw' option is highlighted with a red box. Below it, the '복사' (Copy) option is also highlighted with a red box.

이 값을 SOURCE에 입력하되 index.yaml을 제거한다. 그리고 배포하려면 차트의 이름과 버전을 입력한다.

SOURCE

Repository URL  
<https://github.com/gasbugs/helm-charts/raw/main/stable/>

HELM ▾

Chart  
mychart2 0.1.0

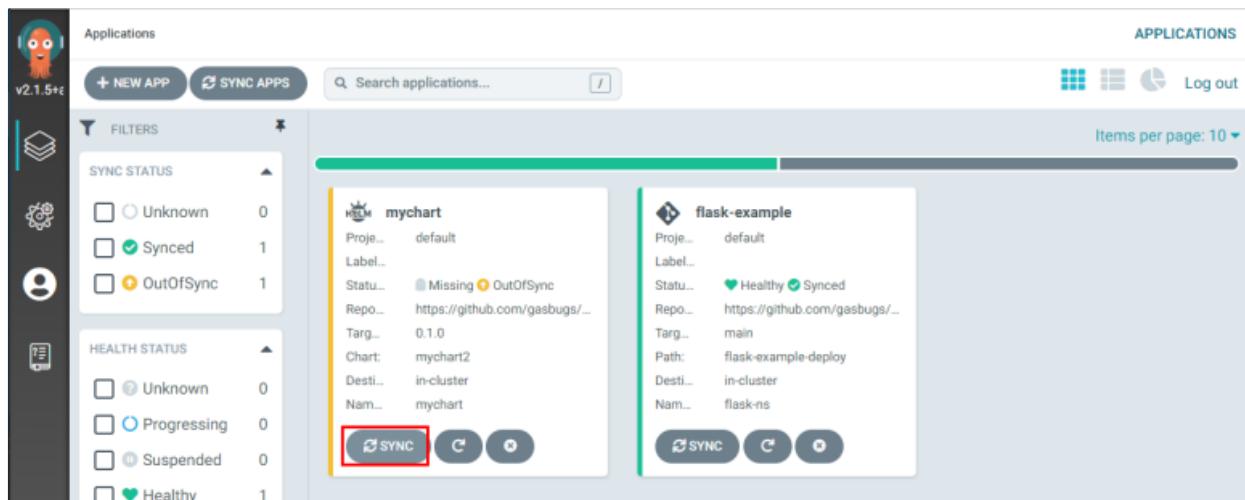
배포하려는 kube-apiserver와 네임스페이스를 입력하자.

DESTINATION

Cluster URL  
<https://kubernetes.default.svc> URL ▾

Namespace  
mychart

모든 구성이 완료되었다면 Create를 누르고 SYNC - SYNCHRONIZE를 사용해 배포를 시작한다.



The screenshot shows the Helm UI interface. On the left, there's a sidebar with icons for Applications, New App, Sync Apps, and Log out. The main area displays two charts: 'mychart' and 'flask-example'. The 'mychart' chart details are as follows:

- Project: default
- Label:
- Status: Missing OutOfSync
- Repo: https://github.com/gasbugs/...
- Target: 0.1.0
- Chart: mychart2
- Destination: in-cluster
- Name: mychart

Below the chart details are three buttons: SYNC (highlighted with a red box), C, and O.

헬름차트로 배포된 앱이 정상적으로 구동중인지 확인해보자!

```
$ kubectl get svc,deploy -n mychart NAME TYPE CLUSTER-IP EXTERNAL-IP PORT(S)
AGE service/mychart-mychart2 ClusterIP 10.8.0.5 <none> 80/TCP 24s NAME READY
UP-TO-DATE AVAILABLE AGE deployment.apps/mychart-mychart2 1/1 1 1 25s
```