

# 쿠버네티스 CI/CD 환경 구성과 활용

- ▶▶ CI/CD 이해
- ▶▶ 도커를 활용한 Jenkins 설치
- ▶▶ 깃헙 Webhook 구성과 젠킨스  
파이프라인 레지스트리인 설정
- ▶▶ Argo를 활용한 CD 환경 구축
- ▶▶ harbor를 활용한 컨테이너 레지스트리 구축



# CI/CD 이해

# CI/CD 이해

## CI/CD의 필요

출처: [www.jetbrains.com](http://www.jetbrains.com)

### ● 기존 개발 프로세스

- 소프트웨어를 출시하기까지는 힘든 과정과 오랜 시간이 필요
- 버그가 발견될 위험을 안고 몇 주의 시간을 들여 수작업으로 통합과 구성 및 테스트 작업을 수행
- 늘 출발점으로 되돌아가야 할 수도 있다는 두려움을 가짐
- 코드 릴리스를 위해서는 오랜 준비 과정이 필요하기 때문에 새로운 릴리스를 내놓기까지 적어도 몇 달은 소요

### ● CI/CD의 개념

- 통합, 전달 및 배포 도구를 의미
- CI/CD의 도움을 받아 많은 조직들이 이미 품질 저하 없이 릴리스 간격을 단축
- CI/CD를 사용하면 반복적 빌드, 테스트 및 배포 작업을 처리하고, 문제가 있을 때 경고
- 자동화된 파이프라인을 통해 코드 변경을 원활하게 진행

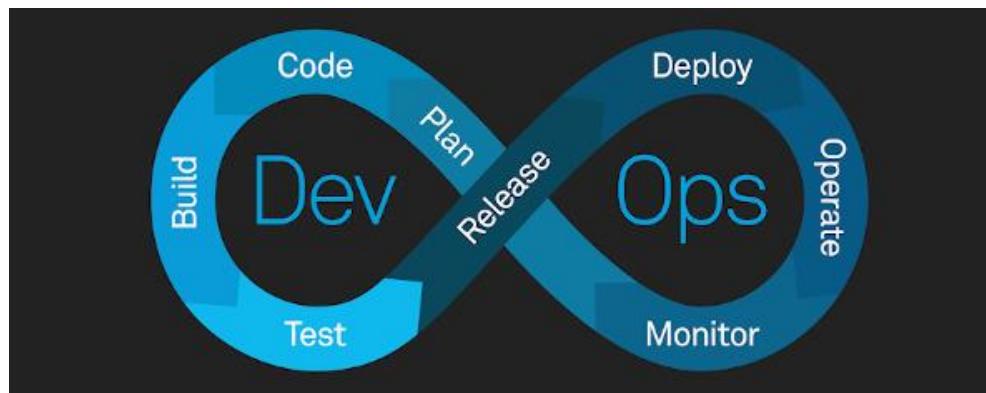


그림 출처: <https://cd.foundation/>

# CI/CD 이해

## CI/CD의 장점

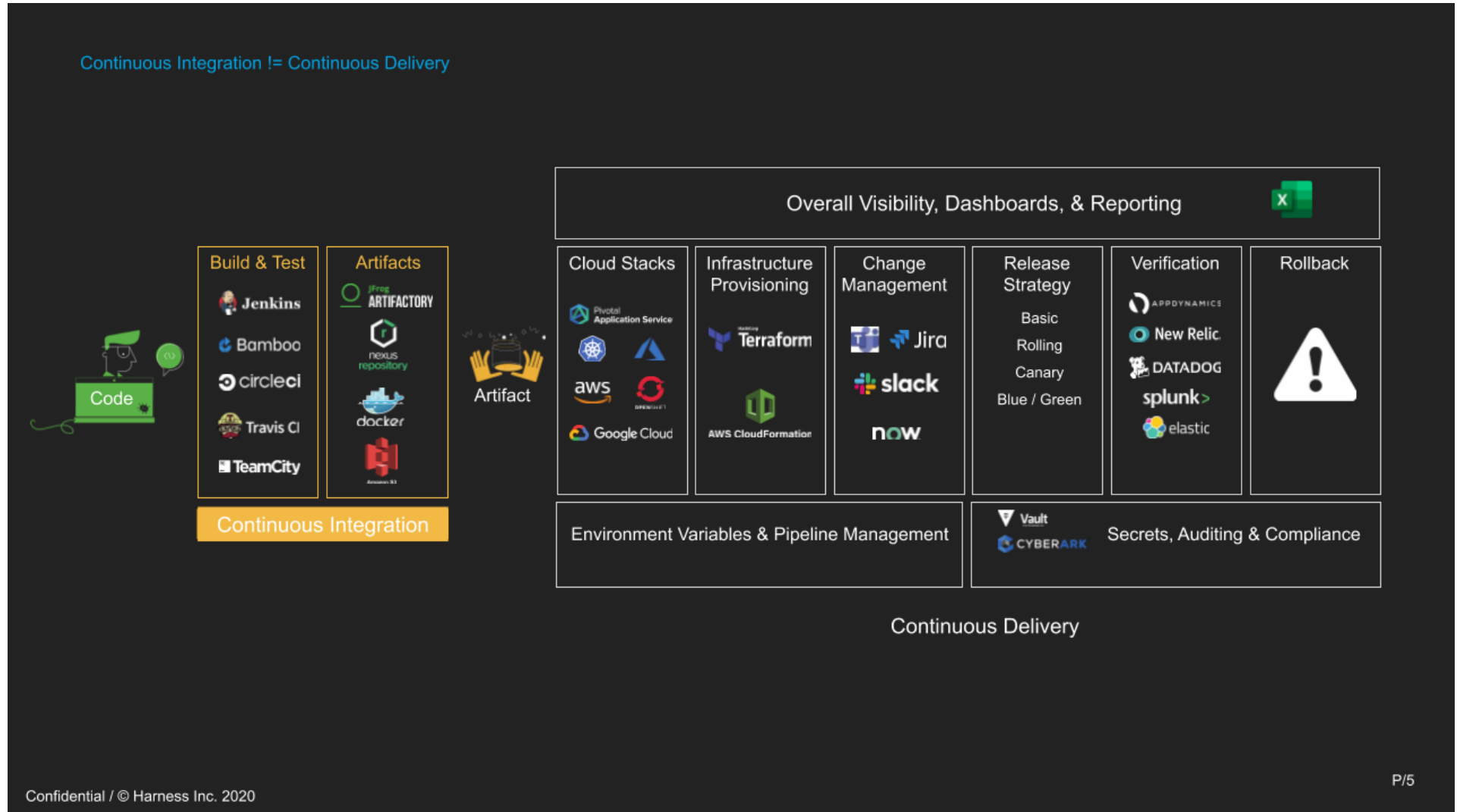
출처: [www.jetbrains.com](http://www.jetbrains.com)

- **시장 출시 기간 단축:** Agile 및 DevOps 기술을 채택하여 개발 프로세스에 혁신을 이루고 사용자에게 지속적인 개선을 제공
  - 위험 감소: 사전 프로덕션 환경에 테스터 또는 실제 사용자를 참여시켜 초기에 자주 사용자와 함께 혁신적 기능을 테스트
  - 검토 시간 단축: 통합 도구를 이용하는 개발자는 자연스럽게 더 자주 코드를 변경하고 싶은 마음이 생기게 되는데, 경험적으로 최소 하루에 한 번은 코드를 변경, 동일한 기반 위에서 작업할 수 있을 뿐만 아니라 코드 검토 속도가 빨라지고 변경 사항을 더 쉽게 통합
- **코드 품질 개선:** CI/CD 파이프라인의 핵심 기능으로, 빌드할 때마다 실행되는 일련의 자동화 테스트
  - 프로덕션 환경으로 원활한 전환: 빌드 자동화, 테스트, 환경 생성 및 배포를 추가하면 각 단계의 일관성과 반복성을 높임, 지속적으로 최적화
  - 더 빠른 버그 수정: 변경 사항을 정기적으로 커밋하고 자주 제공하면 프로덕션으로 전환하는 릴리스 사이에 코드 변경이 상대적으로 적기 때문에 문제의 원인을 찾기가 훨씬 쉬움
  - 효율적인 인프라: 지속적 배포 단계가 더 빠르고 강력해질 뿐만 아니라 개발 작업에 대한 중단을 최소화하면서 추가적인 미리보기 및 교육 환경에 대한 요청에 신속하게 대응
  - 진행상황에 대한 평가 가능: 테스트 커버리지, 결함률, 테스트 수정 시간에 이르는 전체 평가 지표가 제공, 데이터를 확보하면 주의가 필요한 부분을 확인하여 파이프라인을 지속적으로 개선
  - 더 긴밀한 피드백 루프: 지속적 배포 주기마다 통찰력을 적용하면 변경을 수행하는 즉시 그 효과를 확인
  - 협업 및 커뮤니케이션: CI/CD를 시작하려면 팀 사이의 장벽을 허물고 더 많은 의사 소통의 분위기를 마련
- **창의력 극대화:** 컴퓨터를 사용하여 반복적인 작업을 수행함으로써 프로세스를 자동화하면 보다 창의적인 작업에 열중

# CI/CD 이해

## CI/CD에 사용되는 도구들

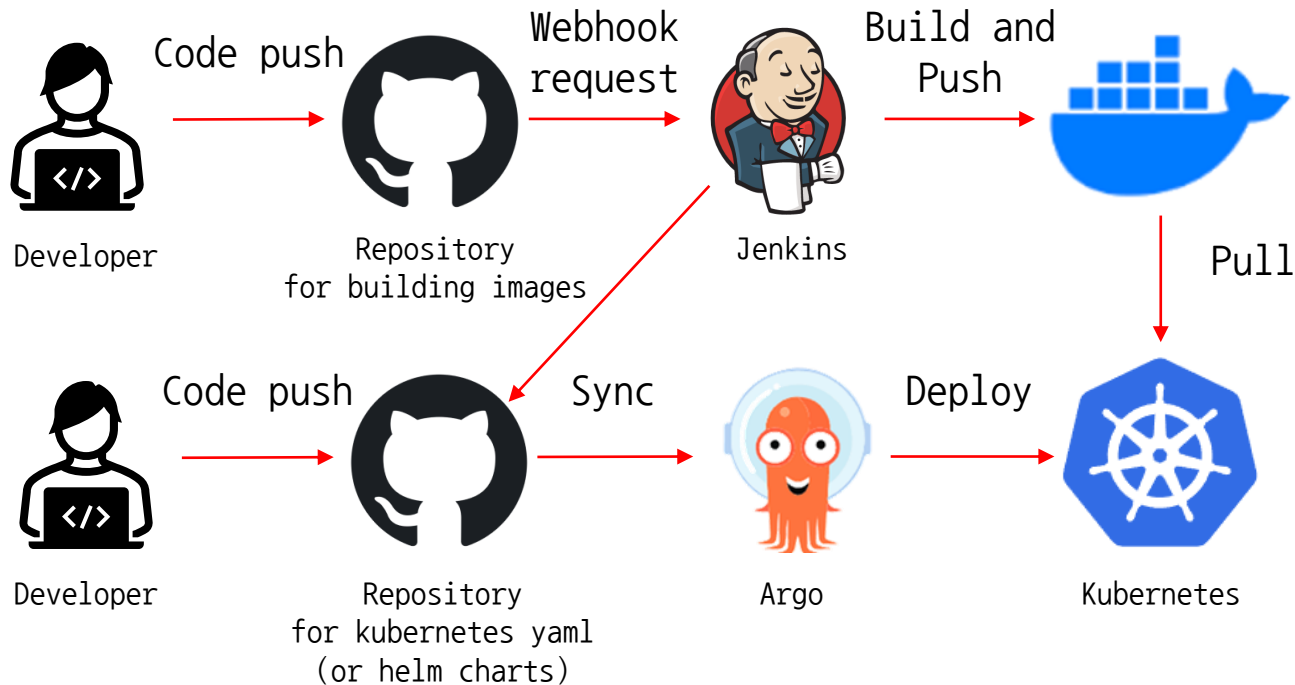
그림 출처: <https://cd.foundation/>



# CI/CD 이해

## 도커와 쿠버네티스 CI/CD 플랫폼

- 개발팀에서 코드를 작성하여 깃헙 레파지토리에 코드 푸시
- 코드 푸시 이벤트를 감지해 Webhook Request를 사용하여 Jenkins에 코드 전송
- Jenkins에서 코드를 사용해 빌드, 테스트한 후 도커 레지스트리에 업로드
- 쿠버네티스 배포를 위한 매니페스트를 위한 두 번째 깃헙 레파지토리 구성
- Argo를 사용해 실시간으로 현재 레파지토리와 클러스터의 상태를 모니터링하고 싱크 상태를 조정



# 도커를 활용한 Jenkins 설치

# 도커를 활용한 Jenkins 설치

## ▶▶▶ 젠킨스란?

- 어떤 규모로든 훌륭한 시스템을 구축하기 위한 도구
- 선도적인 오픈 소스 자동화 서버
- 모든 프로젝트를 빌드, 배포 및 자동화하는 데 지원
- 수백 개의 플러그인 제공



### 지속적인 통합 및 지속적인 납품

확장 가능한 자동화 서버인 Jenkins는 간단한 CI 서버로 사용하거나 모든 프로젝트의 연속 배달 허브로 전환할 수 있습니다.



### 간편한 설치

젠킨스는 독립형 자바 기반 프로그램으로, 윈도우, 리눅스, 맥OS 및 기타 유닉스와 같은 운영 체제에 대한 패키지와 함께 즉시 실행 될 준비가되어 있습니다.



### 쉬운 구성

Jenkins는 웹 인터페이스를 통해 쉽게 설정하고 구성할 수 있으며, 여기에는 즉석 오류 검사 및 기본 제공 도움말이 포함됩니다.



### 플러그인

업데이트 센터에 수백 개의 플러그인을 갖춘 Jenkins는 지속적인 통합 및 지속적인 배달 도구 체인의 거의 모든 도구와 통합됩니다.



### 확장

젠킨스는 플러그인 아키텍처를 통해 확장 될 수 있습니다. 젠킨스 무엇을 할 수 있는 거의 무한 한 가 가능성을 제공.



### 분산

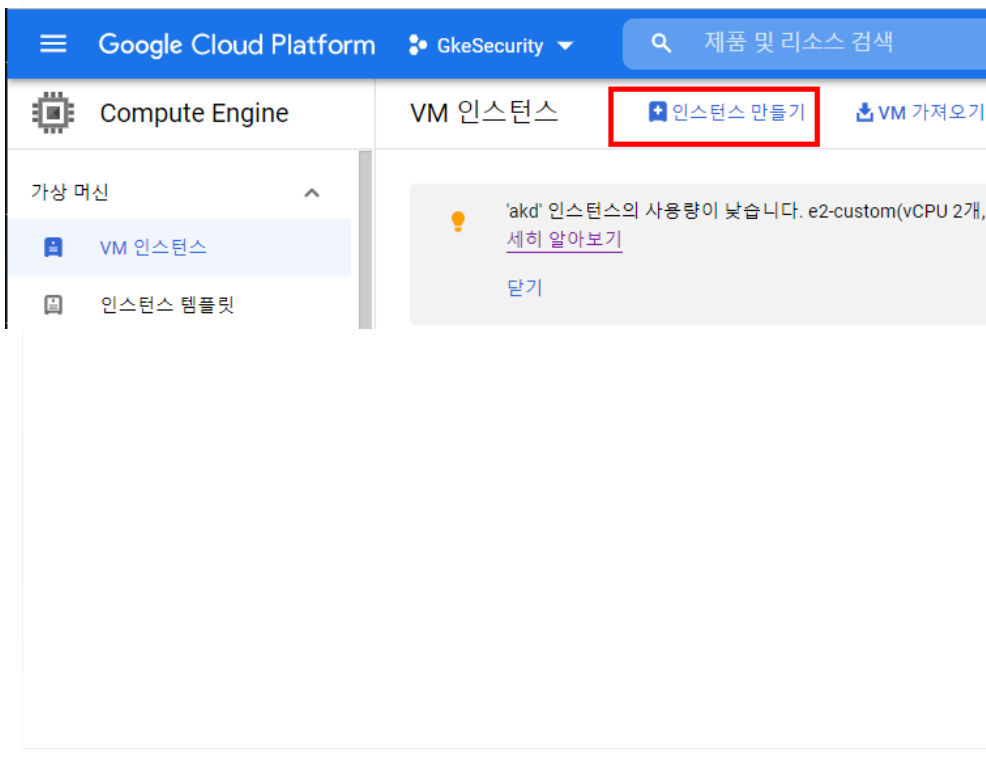
Jenkins는 여러 컴퓨터에 작업을 쉽게 배포할 수 있으며 여러 플랫폼에서 빌드, 테스트 및 배포를 더 빠르게 구축할 수 있습니다.



# 도커를 활용한 Jenkins 설치

## Jenkins 서버 생성

- Jenkins 서버 구성을 시작
- Jenkins의 서버는 외부의 공개된 IP 필요
- 공용 IP를 편하게 구성하기 위해 GCP에 인스턴스를 구성하여 Jenkins를 구축
- GCP에서 인스턴스를 생성
- 인스턴스 이름은 jenkins-ci로 구성하고 CPU는 2개 메모리는 4GB를 선택



# 도커를 활용한 Jenkins 설치

## Jenkins 서버 생성

- 인스턴스의 부팅 디스크에서 변경을 누르고 Ubuntu 20.04 100GB 디스크로 구성
- 방화벽은 뒤에 세팅할 다른 서비스를 위해 HTTP/HTTPS 트래픽을 허용

부팅 디스크 ?

유형	새로운 균형 있는 영구 디스크
크기	100GB
이미지	Ubuntu 20.04 LTS

변경

공개 이미지

커스텀 이미지

스냅샷

기존 디스크

운영체제  
Ubuntu

버전 \*  
Ubuntu 20.04 LTS

amd64 focal image built on 2021-09-27, supports Shielded VM features

부팅 디스크 유형 \*  
균형 있는 영구 디스크

크기(GB) \*  
100

고급 구성 표시

선택

취소

# 도커를 활용한 Jenkins 설치

## ▶ Jenkins 서버 구성

- 빠르고 쉬운 설치를 위해 도커를 설치하고 도커 이미지를 사용해 Jenkins를 배포
- Jenkins를 배포할 때는 일부 디렉토리를 공유하도록 설정
- 도커 소켓 또한 공유하도록 구성
- 이 소켓을 사용해 Jenkins는 호스트에 설치된 도커 기능을 사용

```
# 관리자 권한  
sudo -i
```

```
# docker 설치  
apt update && apt install -y docker.io
```

```
# 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:latest
```

# 도커를 활용한 Jenkins 설치

## Jenkins 서버 구성

- 젠킨스 서비스를 위해 방화벽을 8080 포트 허용

```
gcloud compute firewall-rules create jenkins-ci --allow=tcp:8080
```

- 젠킨스 도커 클라이언트 설치

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

- 젠킨스 패스워드 확인

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

# 도커를 활용한 Jenkins 설치

## ▶ Jenkins 서버 구성

- 젠킨스 설치를 위해 웹으로 접근해 플러그인 설치

The image displays three overlapping screenshots of the Jenkins web interface, illustrating the initial setup process:

- Unlock Jenkins:** The first screenshot shows the 'Getting Started' page with the title 'Unlock Jenkins'. It instructs the user to ensure Jenkins is securely set up by the administrator, mentioning the log file and the location `/var/jenkins_home/secrets/initialAdminPassword`. It asks the user to copy the password from either location and paste it into the 'Administrator password' field.
- Customize Jenkins:** The second screenshot shows the 'Getting Started' page with the title 'Customize Jenkins'. It states that 'Plugins extend Jenkins with additional features to support many different needs.' Below this, there are two buttons: 'Install suggested plugins' (highlighted with a red border) and 'Select plugins to install'. The 'Select plugins to install' button has a sub-instruction: 'Select and install plugins most suitable for your needs.'
- Instance Configuration:** The third screenshot shows the 'Getting Started' page with the title 'Instance Configuration'. It includes a 'Jenkins URL' field with the value `http://34.135.119.94/`. Below the field, it explains that the Jenkins URL is used to provide the root URL for absolute links to various Jenkins resources and is required for the proper operation of many Jenkins features. It also notes that the proposed default value is **not saved yet** and is generated from the current request, if possible.

# 도커를 활용한 Jenkins 설치

## Jenkins 서버 구성

- 젠킨스 구성이 완료되면 Jenkins 관리 - 플러그인 관리로 접근
- docker pipeline을 검색해서 설치 진행

**Jenkins 관리**

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 정보등을 설정합니다.
- Global Tool Configuration: Configure tools, their locations and automatic installers.
- 플러그인 관리**: Jenkins의 기능을 확장하기 위한 플러그인을 추가, 제거, 사용, 미사용으로 설정할 수 있습니다.

노드 관리: Add, re, various

검색: docker pipeline

업데이트된 플러그인 목록 | 설치 가능 | 설치된 플러그인 목록 | 고급

Install ↑	Name	Version	Release
<input checked="" type="checkbox"/>	<b>Docker Pipeline</b> Deployment   DevOps   docker   pipeline Build and use Docker containers from pipelines.	1.26	8 months ago

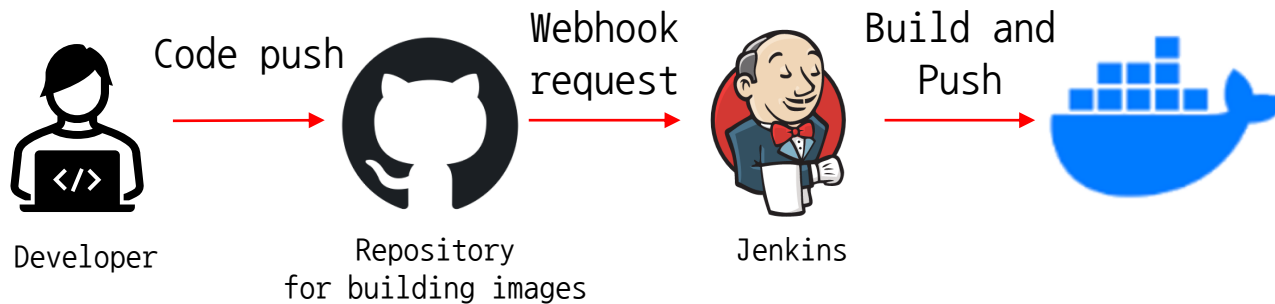
**Install without restart** | Download now and install after restart | Update information obtained: 11 min ago | 지금 확인

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## github 레파지토리 구성

- 필자가 미리 구성해둔 flask-example 프로젝트를 활용
- <https://github.com/gasbugs/flask-example>
- 이 프로젝트를 fork해서 내 레파지토리로 가져와야 함
- 이 예제에는 Docker 이미지를 구성하는 Dockerfile과 Jenkins 파이프라인에 사용할 Jenkinsfile이 함께 첨부

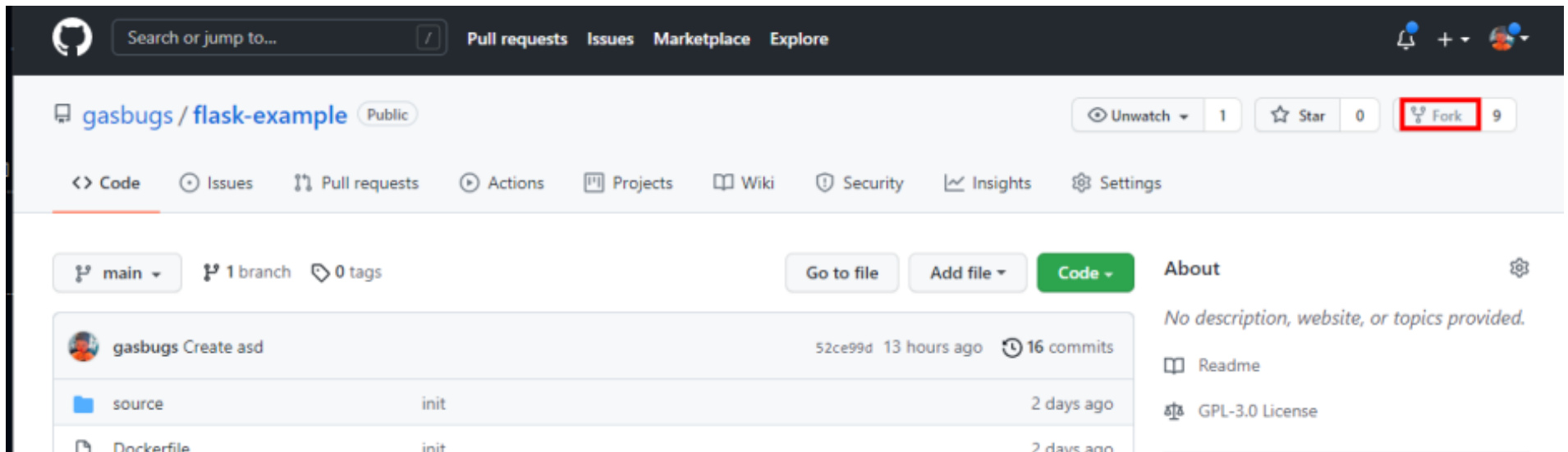




# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## github 레파지토리

- github 레파지토리를 구성
- 필자가 미리 구성해둔 flask-example 프로젝트를 활용
- <https://github.com/gasbugs/flask-example>
- 이 프로젝트를 fork해서 내 레파지토리로 가져와야 함
- 이 예제에는 Docker 이미지를 구성하는 Dockerfile과 Jenkins 파이프라인에 사용할 Jenkinsfile이 함께 첨부



# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## github 레파지토리 웹훅 설정

- 변경사항이 있을 때마다 jenkins에 알릴 수 있도록 Webhook을 설정
- 레파지토리의 Settings - Webhooks에 구축된 젠킨스 인스턴스 URL 주소와 /github-webhook을 함께 작성하고 저장

gasbugs / flask-example Public

Unwatch 1 Star 0 Fork 9

<> Code Issues Pull requests Actions Projects Wiki Security Insights Settings

Options

Manage access

Security & analysis

Branches

Webhooks

Notifications

Integrations

Deploy keys

Actions

Environments

Secrets

Pages

### 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 \*

http://35.193.232.22/github-webhook/

Content type

application/json

Secret

Which events would you like to trigger this webhook?

☒ Just the push event.

☐ Send me everything.

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## ▶ Jenkins에서 파이프라인 구성

- Jenkins에서 Webhook 이벤트를 받아 빌드를 시작할 수 있도록 구성
- 새로운 Item을 클릭
- 이름을 적절하게 구성하고 pipeline을 선택
- 이름: flask-example-docker-pipeline



The screenshot shows the Jenkins Dashboard. On the left sidebar, the '새로운 Item' (New Item) button is highlighted with a red box. The main area displays the 'Enter an item name' form with the text 'flask-example-docker-pipeline' entered. Below the form, there are two options: 'Freestyle project' and 'Pipeline'. The 'Pipeline' option is highlighted with a red box. The 'Pipeline' option includes 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.'

**Jenkins**

Dashboard ▶

새로운 Item

사람

빌드 기록

**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.

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## Jenkins에서 파이프라인 구성

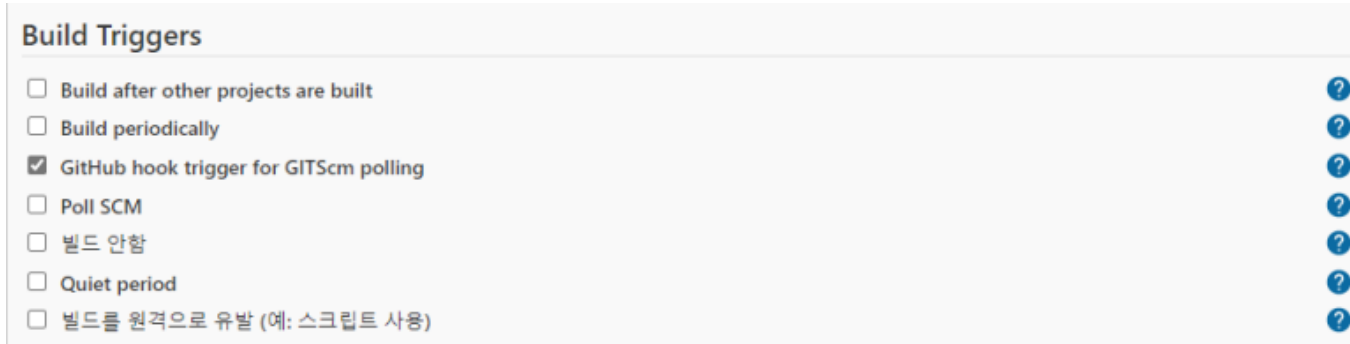
- General에는 Github프로젝트를 선택하고 프로젝트 URL 정보를 입력
- 포크한 본인의 레파지토리를 입력

The screenshot shows the Jenkins 'General' configuration page. The 'General' tab is active, with other tabs like 'Build Triggers', 'Advanced Project Options', and 'Pipeline' visible. The '설명' (Description) field is empty. Below it, the '[Plain text] 미리보기' (Preview) section shows a list of checkboxes: 'Do not allow concurrent builds', 'Do not allow the pipeline to resume if the controller restarts', and 'GitHub project' (which is checked). The 'Project url' field is filled with 'https://github.com/gasbugs/flask-example'. At the bottom, there are more unchecked checkboxes: 'Pipeline speed/durability override', 'Preserve stashes from completed builds', 'Throttle builds', '오래된 빌드 삭제' (Delete old builds), and '이 빌드는 매개변수가 있습니다' (This build has parameters). A '고급...' (Advanced...) button is located on the right side of the configuration area.

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## Jenkins에서 파이프라인 구성

- 빌드 트리거는 Webhook을 통해서 시작할 수 있도록 설정



The screenshot shows the 'Build Triggers' section of the Jenkins configuration interface. It contains a list of checkboxes for different build triggers. The 'GitHub hook trigger for GITScm polling' option is checked, while others like 'Build after other projects are built', 'Build periodically', 'Poll SCM', '빌드 안함', 'Quiet period', and '빌드를 원격으로 유발 (예: 스크립트 사용)' are unchecked. Each checkbox has a corresponding help icon (a question mark in a blue circle) to its right.

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 설정
- Credentials는 Add 버튼을 눌러 새로 생성



The screenshot shows the 'Pipeline' configuration section in Jenkins. Under the 'Definition' tab, 'Pipeline script from SCM' is selected. In the 'SCM' section, 'Git' is chosen. Under 'Repositories', the 'Repository URL' is set to 'https://github.com/gasbugs/flask-example'. In the 'Credentials' section, a dropdown menu shows '- none -' and an 'Add' button with a key icon, which is highlighted with a red rectangle.

Pipeline

Definition

Pipeline script from SCM

SCM

Git

Repositories

Repository URL

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

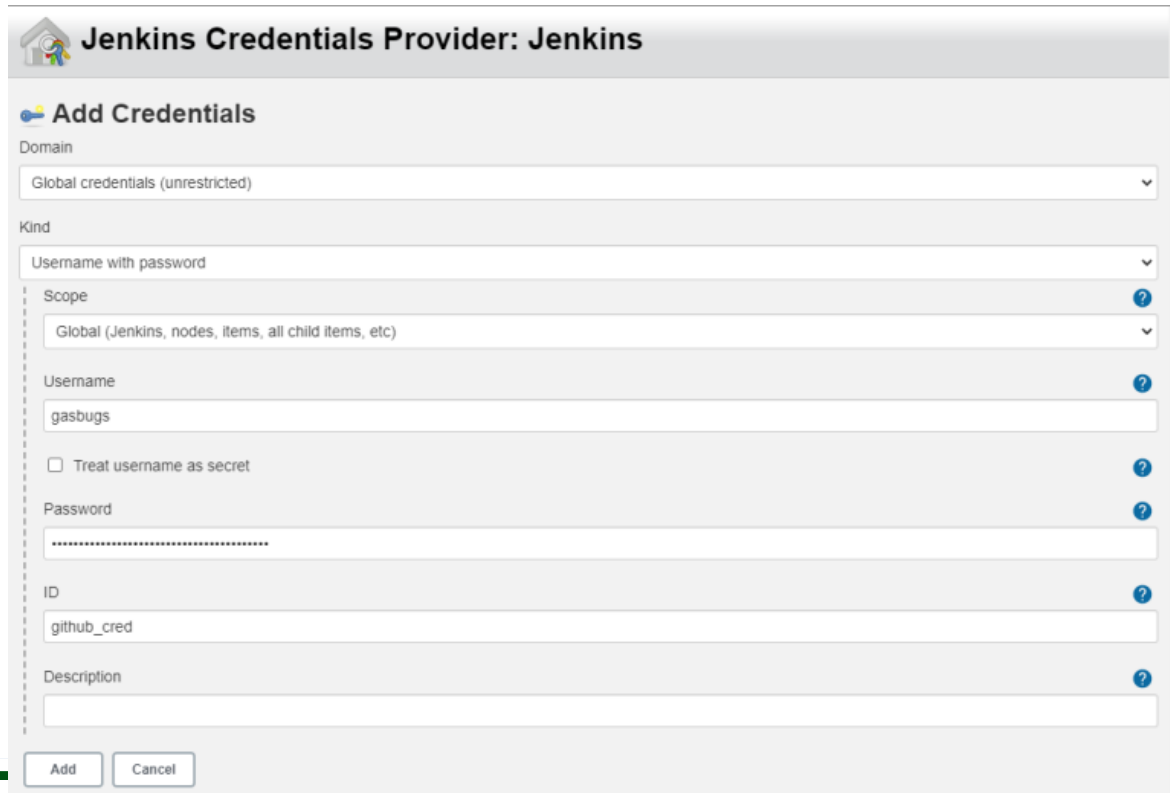
Credentials

- none - Add

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## Jenkins에서 파이프라인 구성

- 깃헙에서 토큰을 발급 받아 다음과 같이 입력
- 토큰 발급에 대한 상세 절차는 생략
  - username: github ID
  - password: Token 값
  - github\_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

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## ▶ Jenkins에서 파이프라인 구성

- main 정보를 입력하고 Jenkinsfile에 대한 위치 정보는 그대로 유지
- Flask-example/Jenkinsfile의 파이프라인 정보를 읽어서 빌드 절차를 진행
- 모든 것이 완료되면 “저장” 버튼

Branches to build

Branch Specifier (blank for 'any')

\*/main

Add Branch

Repository browser

(자동)

Additional Behaviours

Add

Script Path

Jenkinsfile

☒ Lightweight checkout

Pipeline Syntax

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## Jenkins에 도커 허브 인증 정보 입력

- 이미지 빌드를 수행한 뒤 docker hub로 푸시를 진행하려면 docker hub에 대한 인증정보가 필요
- Jenkins 관리에 Manage Credentials로 진입

The screenshot shows the Jenkins Dashboard. On the left sidebar, the 'Jenkins 관리' (Manage Jenkins) link is highlighted with a red box. The main content area is divided into two sections: 'System Configuration' and 'Security'. In the 'System Configuration' section, there are links for '시스템 설정' (System Configuration) and 'Global Tool Configuration'. In the 'Security' section, there are links for 'Configure Global Security' and 'Manage Credentials', with the latter being highlighted by a red box. A yellow warning banner at the top right states: 'building on the controller node can be a security issue. you should set up distributed builds. see the documentation.'



# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## Jenkins에 도커 허브 인증 정보 입력

- Jenkins 스토어로 진입한 후에 Global credentials로 계속 접근

### Stores scoped to Jenkins

The screenshot shows the Jenkins web interface. At the top, the 'Stores scoped to Jenkins' section is visible, with a table containing a 'Jenkins' store. Below this, the 'System' section shows a table with 'Global credentials (unrestricted)'. A red box highlights the 'Jenkins' store, and a red arrow points down to the 'Global credentials (unrestricted)' entry. Another red box highlights the 'Global credentials (unrestricted)' entry, and a red arrow points down to the 'Add Credentials' button in the left sidebar of the 'Global credentials (unrestricted)' page. The 'Add Credentials' button is also highlighted with a red box.

Store	Domains
Jenkins	(global)

Domain	Description
Global credentials (unrestricted)	Credentials that should be available irrespective of domain specification to requirements matching.

아이콘: S M L

**Jenkins**

Dashboard > Credentials > System > Global credentials (unrestricted)

[Back to credential domains](#)

[Add Credentials](#)

### Global credentials (unrestricted)

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

ID	Name	Kind
github_cred	gasbugs/*****	Username with password

아이콘: S M L

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## ▶ Jenkins에 도커 허브 인증 정보 입력

- username과 password에 도커 허브의 ID와 패스워드를 차례로 입력
- IID는 반드시 docker-hub로 입력
- Jenkinsfile에서 이 ID를 참조하도록 구성

The screenshot shows the Jenkins configuration page for a new credential. The 'Kind' dropdown is set to 'Username with password'. The 'Scope' dropdown is set to 'Global (Jenkins, nodes, items, all child items, etc)'. The 'Username' field contains 'gasbugs'. The 'Treat username as secret' checkbox is unchecked. The 'Password' field is masked with dots. The 'ID' field contains 'docker-hub', which is highlighted with a red rectangle. Each field has a help icon (question mark) to its right.

Kind

Username with password

Scope

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

Username

gasbugs

☐ Treat username as secret

Password

.....

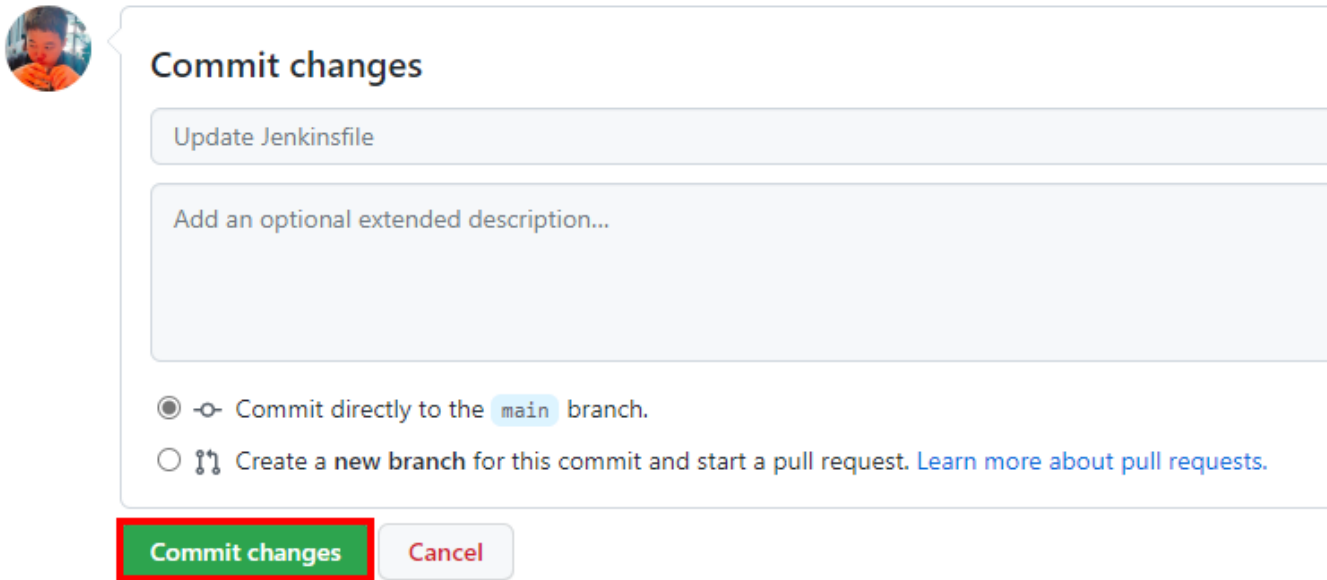
ID

docker-hub

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## 빌드와 푸시를 위한 Jenkinsfile 설정 변경

- 코드를 변경해서 Github의 웹훅이 적절히 전달되는지 확인
- Github 사이트에서 flask-example/Jenkinsfile를 선택하고 수정 버튼을 누르면 푸시 이벤트 발생
- gasbugs라는 필자의 ID를 당신의 docker hub ID로 변경하고 수정 완료



A screenshot of the GitHub 'Commit changes' dialog. On the left is a circular profile picture of a person with dark hair. The dialog box has a title 'Commit changes'. Below the title is a text input field containing 'Update Jenkinsfile'. Underneath is a larger text area with the placeholder 'Add an optional extended description...'. At the bottom, there are two radio button options: the first is selected and says 'Commit directly to the main branch.', the second is unselected and says 'Create a new branch for this commit and start a pull request. Learn more about pull requests.' At the very bottom are two buttons: 'Commit changes' (highlighted with a red border) and 'Cancel'.

# 깃헙 레파지토리 Webhook 구성과 젠킨스 파이프라인 설정

## Github 웹훅 테스트

- 코드를 바꿨지만 젠킨스로 가보면 빌드가 진행되지 않음
- 첫 빌드는 jenkins에서 Build now 버튼을 눌러 직접 진행 주어야 함
- 첫 빌드 후에는 푸시 이벤트 발생 시 자동으로 빌드 진행

**Jenkins** Dashboard > flask-example-docker-pipeline

Back to Dashboard  
Status  
Changes  
**Build Now**  
구성  
Pipeline 삭제  
Full Stage View  
GitHub  
Rename  
Pipeline Syntax  
GitHub Hook Log  
Build History

### Pipeline flask-example-docker-pipeline

상세 내용 열람  
프로젝트 중지하기

#### Stage View

Average stage times:  
(Average full run time: ~1min 3s)

	Clone repository	Build image	Push image	Build image	Push image
Average	904ms	21s	7s	0ms	0ms
Oct 23 11:53	904ms	42s	9s	390ms	4s

#### 고정링크

• Last build, (#1), 26 sec 전

# Argo를 활용한 CD 환경 구축

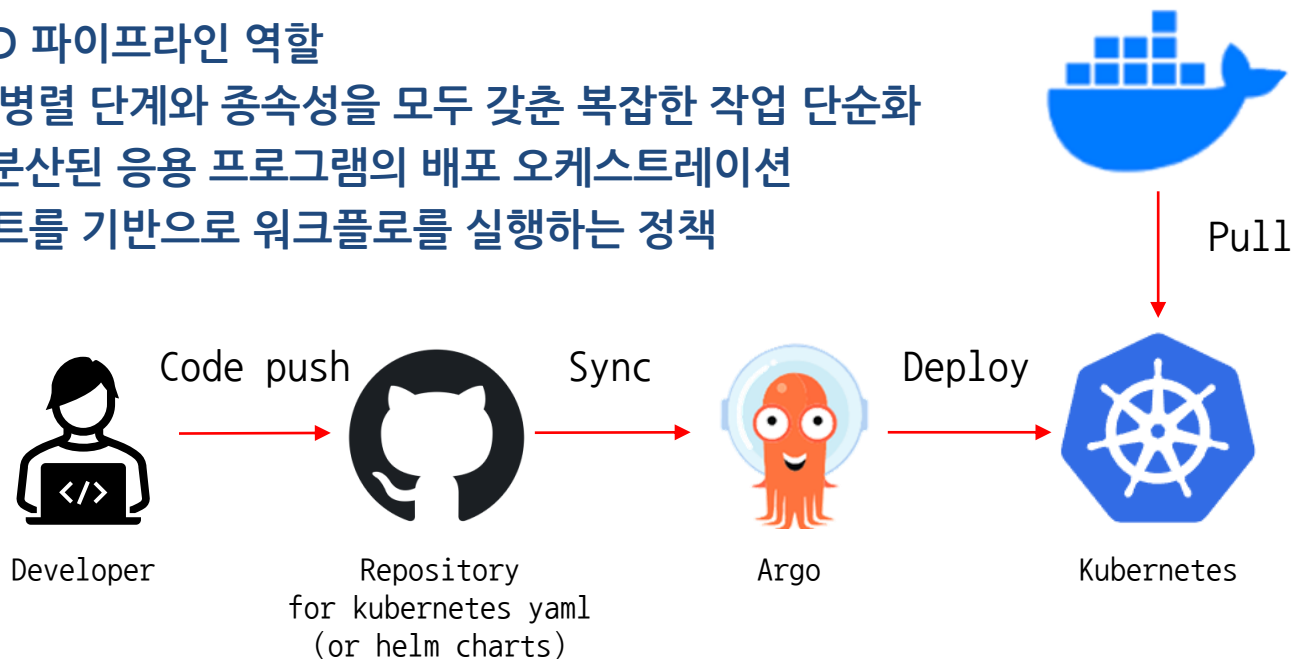
# Argo를 활용한 CD 환경 구축

## argo의 소개

- argo는 Kubernetes용 오픈 소스 도구로 워크플로를 실행하고 클러스터를 관리 GitOps를 수행
- 쿠버네티스의 CD를 담당하는 도구로 쿠버네티스의 컨테이너 네이티브 워크플로우 엔진
- Kubernetes에서 복잡한 워크플로 및 응용 프로그램의 실행을 쉽게 지정하고 예약, 조정

## argo의 워크플로의 역할

- 기존 CI/CD 파이프라인 역할
- 순차적 및 병렬 단계와 종속성을 모두 갖춘 복잡한 작업 단순화
- 복잡하고 분산된 응용 프로그램의 배포 오케스트레이션
- 시간/이벤트를 기반으로 워크플로를 실행하는 정책

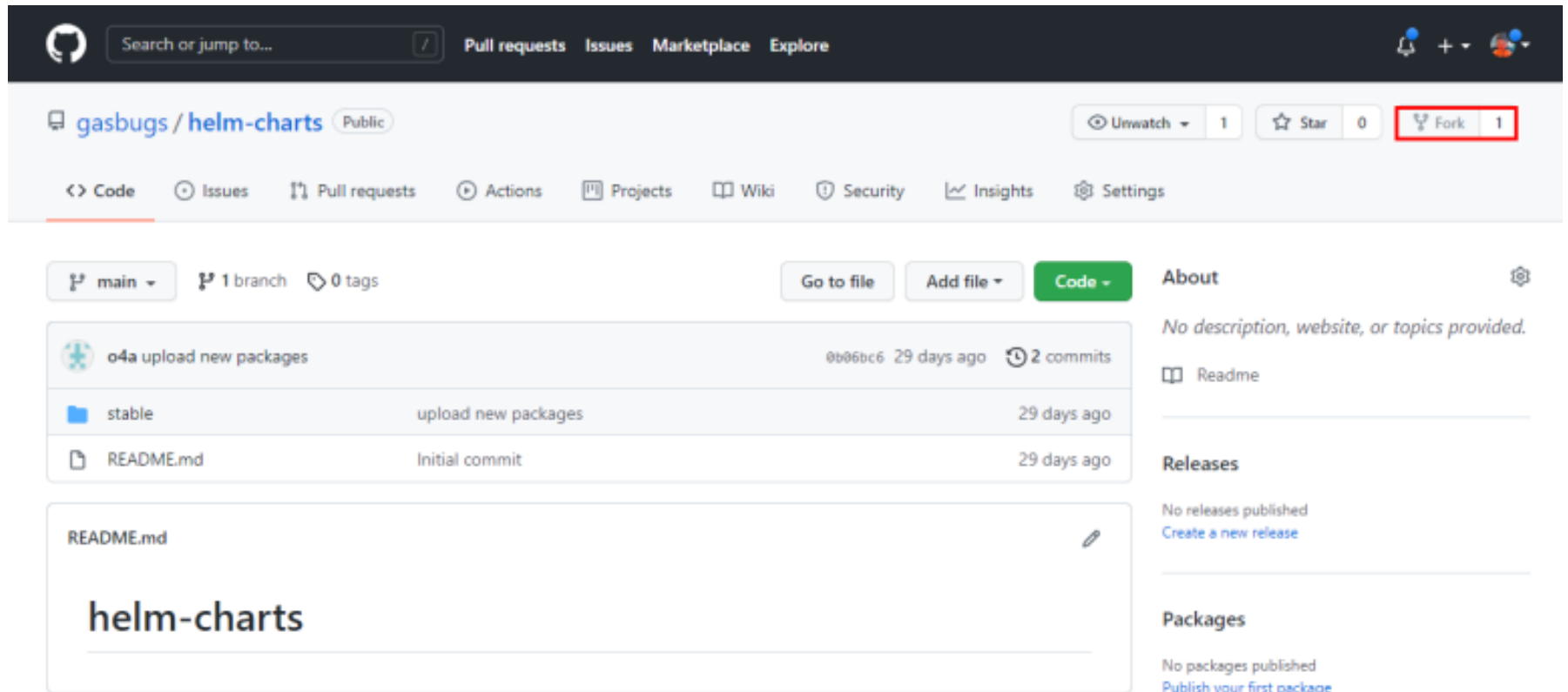


# Argo를 활용한 CD 환경 구축

## github 레파지토리 포크

### ● 깃헙 레파지토리를 구성

- 필자가 미리구성해 둔 레파지토리를 포크
- <https://github.com/gasbugs/flask-example-apps>
- <https://github.com/gasbugs/helm-charts>



The screenshot displays the GitHub interface for the repository 'gasbugs / helm-charts'. At the top, there's a search bar and navigation links for Pull requests, Issues, Marketplace, and Explore. The repository name 'gasbugs / helm-charts' is shown with a 'Public' label. To the right, there are buttons for 'Unwatch', 'Star' (0), and 'Fork' (1), with the 'Fork' button highlighted by a red rectangle. Below this, a navigation bar includes links for Code, Issues, Pull requests, Actions, Projects, Wiki, Security, Insights, and Settings. The main content area shows the repository's structure: a 'main' branch with 1 branch and 0 tags. A table lists the files: 'stable' (upload new packages, 29 days ago) and 'README.md' (Initial commit, 29 days ago). The 'README.md' content is visible, showing the title 'helm-charts'.

# Argo를 활용한 CD 환경 구축

## Argo 설치

- 로드밸런서로 변경하고 서비스의 IP를 확인
- 변경되는데 1분 정도 소요

```
kubectl patch svc argocd-server -n argocd -p '{"spec": {"type": "LoadBalancer"}}'
kubectl get svc argocd-server -n argocd -w
```

// 출력

NAME	TYPE	CLUSTER-IP	EXTERNAL-IP	PORT(S)	AGE
argocd-server	LoadBalancer	10.8.9.12	<pending>	80:32524/TCP,443:31837/TCP	35m
argocd-server	LoadBalancer	10.8.9.12	35.222.254.32	80:32524/TCP,443:31837/TCP	36m

- 시크릿 정보를 통해 argo 패스워드 확인

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



# Argo를 활용한 CD 환경 구축

## Argo 접속

- 로드밸런서로 할당 받은 IP로 https로 접속
- username: admin
- password: <secret을 통해 확인한 값>

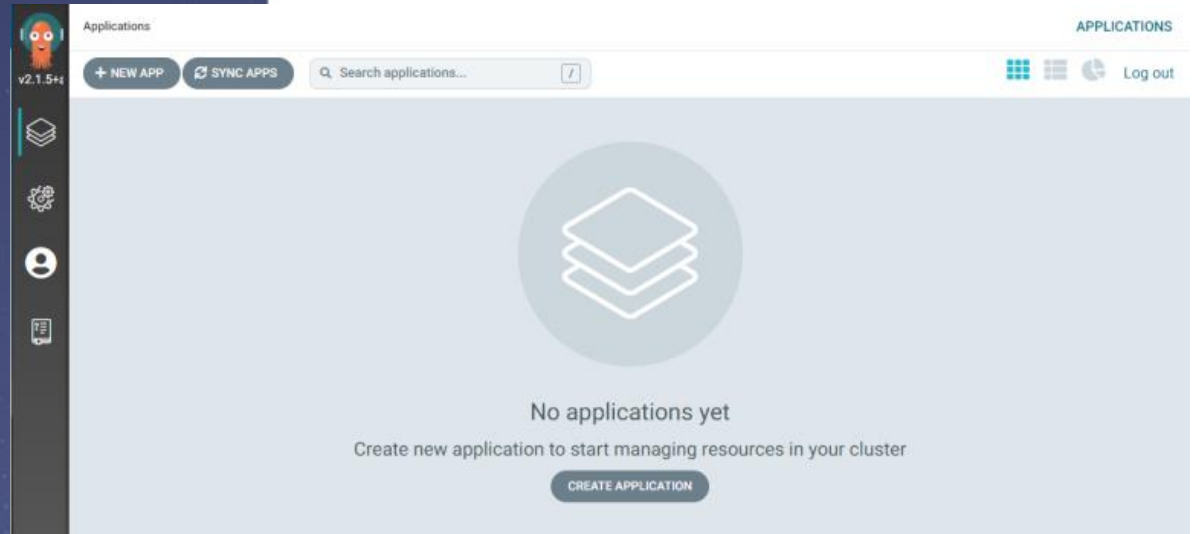
Let's get stuff deployed!



argo

Username

Password



# Argo를 활용한 CD 환경 구축

## Argo 앱 생성

- 애플리케이션 이름: flask-example
- 프로젝트: default (Argo의 기본 프로젝트)
- 싱크 방식: Manual
  - 자동 동기화: Git에 설정된 매니페스트 내용과 현재 애플리케이션의 상태가 다르거나 정상 동작하지 않으면 자동으로 동기화해 애플리케이션을 다시 배포하거나 재구성하는 기능
- AUTO CREATE NAMESPACE: Enable

The screenshot shows the 'GENERAL' tab of the Argo CD application configuration form. The 'Application Name' is 'flask-example' and the 'Project' is 'default'. Under 'SYNC POLICY', 'Manual' is selected. In the 'SYNC OPTIONS' section, 'SKIP SCHEMA VALIDATION' and 'PRUNE LAST' are unchecked, while 'AUTO-CREATE NAMESPACE' is checked. 'APPLY OUT OF SYNC ONLY' is also unchecked. The 'PRUNE PROPAGATION POLICY' is set to 'foreground'. At the bottom, 'REPLACE' is unchecked with a warning icon. An 'EDIT AS YAML' button is in the top right corner.

GENERAL	
Application Name	flask-example
Project	default
SYNC POLICY	
Manual	
SYNC OPTIONS	
<input type="checkbox"/> SKIP SCHEMA VALIDATION	<input checked="" type="checkbox"/> AUTO-CREATE NAMESPACE
<input type="checkbox"/> PRUNE LAST	<input type="checkbox"/> APPLY OUT OF SYNC ONLY
PRUNE PROPAGATION POLICY: foreground	
<input type="checkbox"/> REPLACE ⚠	

# Argo를 활용한 CD 환경 구축

## Argo 앱 생성

- 매니페스트 파일의 소스의 위치를 지정
- flask-example-apps 프로젝트를 링크를 전달
- main 브랜치를 사용하도록 설정
- Path는 배포할 yaml 파일이 있는 디렉토리를 지정
- 우리의 프로젝트에는 flask-example-deploy 아래에 필요한 yaml이 구성

The image shows a screenshot of the Argo CD application configuration form. The form is titled 'SOURCE' and contains three input fields: 'Repository URL', 'Revision', and 'Path'. The 'Repository URL' field is filled with 'https://github.com/gasbugs/flask-example-apps' and has a dropdown menu set to 'GIT'. The 'Revision' field is filled with 'main' and has a dropdown menu set to 'Branches'. The 'Path' field is filled with 'flask-example-deploy'.

SOURCE	
Repository URL	https://github.com/gasbugs/flask-example-apps
Revision	main
Path	flask-example-deploy

# Argo를 활용한 CD 환경 구축

## Argo 앱 생성

- 쿠버네티스 API 서버에 대한 정보를 입력
- 우리가 구성한 argo는 쿠버네티스 내에 구성
- kube-apiserver에 대한 URL 정보를 도메인 주소로 입력
- 네임스페이스는 애플리케이션을 배포할 공간을 의미
- jenkins-ns 네임스페이스에 배포

DESTINATION

Cluster URL

https://kubernetes.default.svc URL ▼

Namespace

flask-ns

# Argo를 활용한 CD 환경 구축

## Argo 앱 싱크

- 모든 설정이 완료되면 상단에 CREATE 버튼을 클릭
- 새로 생성된 app인 flask-example을 확인
- SYNC 버튼을 눌러서 배포 진행

The screenshot displays the Argo CD web interface. On the left, the 'Applications' sidebar shows a list of applications with filters for SYNC STATUS and HEALTH STATUS. The main panel shows the details for the 'flask-example' application, which is currently 'OutOfSync'. A 'SYNC' button is visible at the bottom of the application details. Overlaid on the right is a 'Synchronize' dialog box. This dialog has a 'SYNCHRONIZE' button (highlighted with a red box) and a 'CANCEL' button. It shows the source repository as 'https://github.com/gasbugs/flask-example-apps' and the revision as 'main'. Below this, there are checkboxes for 'PRUNE', 'DRY RUN', 'APPLY ONLY', and 'FORCE'. The 'SYNC OPTIONS' section includes 'SKIP SCHEMA VALIDATION', 'PRUNE LAST', 'AUTO-CREATE NAMESPACE' (checked), and 'APPLY OUT OF SYNC ONLY'. The 'PRUNE PROPAGATION POLICY' is set to 'foreground'. At the bottom, the 'SYNCHRONIZE RESOURCES' section shows two checked items: '/SERVICE/FLASK-NS/FLASK' and 'APPS/DEPLOYMENT/FLASK-NS/FLASK'.

# Argo를 활용한 CD 환경 구축

## ▶ 헬름차트를 활용한 배포

- 앞서 구성한 방법과 동일하게 애플리케이션 이름과 몇 정보를 입력

GENERAL

EDIT AS YAML

Application Name  
mychart

Project  
default

SYNC POLICY  
Manual

SYNC OPTIONS

☐ SKIP SCHEMA VALIDATION

☒ AUTO-CREATE NAMESPACE

☐ PRUNE LAST

☐ APPLY OUT OF SYNC ONLY

PRUNE PROPAGATION POLICY: foreground

☐ REPLACE ⚠

# Argo를 활용한 CD 환경 구축

## ▶ 헬름차트를 활용한 배포

- Github의 helm-charts 레파지토리에서 index.yaml 파일의 링크 확인

The screenshot shows the GitHub repository page for `gasbugs/helm-charts`. The file `helm-charts/stable/index.yaml` is selected. The 'Raw' button in the file toolbar is highlighted with a red box. A red arrow points from this button to the '링크 복사' (Copy link) option in the context menu that appears. The context menu also includes options like '새 탭에서 링크 열기' (Open link in new tab), '새 창에서 링크 열기' (Open link in new window), 'InPrivate 창에서 링크 열기' (Open link in InPrivate window), '장치에 링크 보내기' (Send link to device), '(으)로 링크 저장' (Save link as...), '선택선에 추가' (Add to selection), '공유' (Share), and '웹 캡처' (Web capture).

**SOURCE**

Repository URL

`https://github.com/gasbugs/helm-charts/raw/main/stable/`

Chart

`mychart2`

0.1.0

```
13 version: 0.1.0
14 mychart2:
15   - apiVersion: v2
```

# Argo를 활용한 CD 환경 구축

## ▶ 헬름차트를 활용한 배포

- 앞서 작성한 것과 동일하게 쿠버네티스 API의 도메인과 배포할 네임스페이스 정보 입력

DESTINATION


Cluster URL

https://kubernetes.default.svc

URL ▾



Namespace

mychart

 mychart

Proje... default

Label...

Statu...  Missing  OutOfSync


Repo... https://github.com/gasbugs/...


Targ... 0.1.0


Chart: mychart2


Desti... in-cluster

Nam... mychart

 SYNC







 flask-example

Proje... default

Label...

Statu...  Healthy  Synced


Repo... https://github.com/gasbugs/...


Targ... main


Path: flask-example-deploy

Desti... in-cluster

Nam... flask-ns

 SYNC







# harbor를 활용한 컨테이너 레지스트리 구축

# harbor를 활용한 컨테이너 레지스트리 구축

## Harbor란?

- CNCF 졸업 프로젝트로 오픈 소스 레지스트리
- 정책 및 역할 기반 액세스 제어를 제공
- 아티팩트를 보호하고, 이미지가 스캔, 취약성 확인, 이미지를 신뢰할 수 있도록 서명
- 웹 대시보드 제공




# harbor를 활용한 컨테이너 레지스트리 구축

## Harbor를 설치할 인스턴스 생성

- GCP를 사용해 우분투 인스턴스를 구성, 디스크는 100GB

### 부팅 디스크 ?

유형	새로운 균형 있는 영구 디스크
크기	100GB
이미지	 Ubuntu 20.04 LTS

변경

### 방화벽 ?

태그 및 방화벽 규칙을 추가하여 인터넷에서 들어오는 특정 네트워크 트래픽을 허용합니다.

- ☒ HTTP 트래픽 허용
- ☒ HTTPS 트래픽 허용

# harbor를 활용한 컨테이너 레지스트리 구축

## docker-compose를 사용해서 배포

- 도커 설치가 필요
- 관리자 권한으로 넘어와서 도커 설치를 진행

```
sudo -i
```

```
# 도커 설치
```

```
apt update && apt install docker.io -y
```

```
# 도커 컴포즈 설치
```

```
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
```

# harbor를 활용한 컨테이너 레지스트리 구축

## Harbor 설치

- harbor를 설치하기 위해 harbor 사이트에서 명령을 가져와 셸스크립트를 다운로드하고 실행

```
wget
https://gist.githubusercontent.com/kacole2/95e83ac84fec950b1a70b0853d6594dc/raw/ad6d65
d66134b3f40900fa30f5a884879c5ca5f9/harbor.sh
bash harbor.sh
```

- 설치를 시작하면 IP와 도메인을 선택하는 메뉴가 나오는데 IP를 사용할 예정이므로 1번을 선택

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

# harbor를 활용한 컨테이너 레지스트리 구축

## Harbor 설치

- HTTPS 통신을 위한 인증서를 구성하고 설정하여 설치를 진행
- 다음 스크립트를 실행해 CA와 Harbor에서 사용할 Certificate를 생성

```
#!/bash
#
mkdir pki
cd pki

# ca 키와 인증서 생성
openssl req -x509 -nodes -days 3650 -newkey 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
```

# harbor를 활용한 컨테이너 레지스트리 구축

## Harbor 설치

- harbor.yml 템플릿을 사용해 harbor의 설정을 구성

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

- harbor.yml 파일의 내부에서 호스트 이름과 키, 인증서의 경로만 바꿈
- 여기서 초기 패스워드와 유저 아이디도 확인 가능
- 호스트 이름의 IP는 현재 인스턴스의 공인 IP를 입력

```
hostname: 34.134.229.160
```

<중략>

```
# 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/server.crt
```

```
private_key: /etc/docker/certs.d/server/server.key
```

# harbor를 활용한 컨테이너 레지스트리 구축

## ▶ Harbor 설치

- 끝으로 준비 및 설치 스크립트를 실행
- 준비 스크립트는 이미지를 준비하고 인증서 파일을 위한 설정을 구성
- install.sh 파일은 도커 컴포즈를 사용해 harbor 실행에 필요한 컨테이너들을 배포

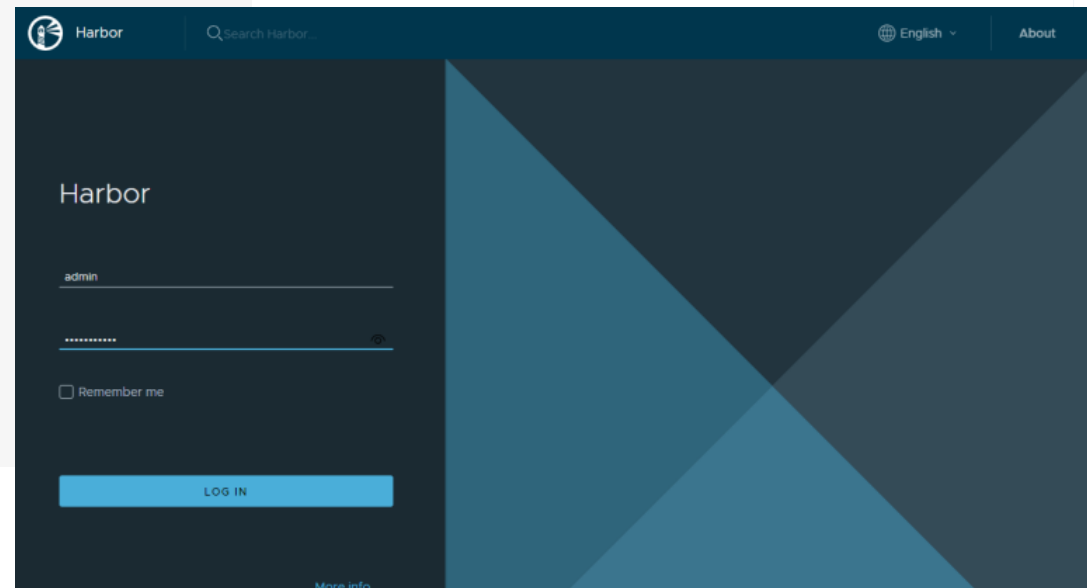
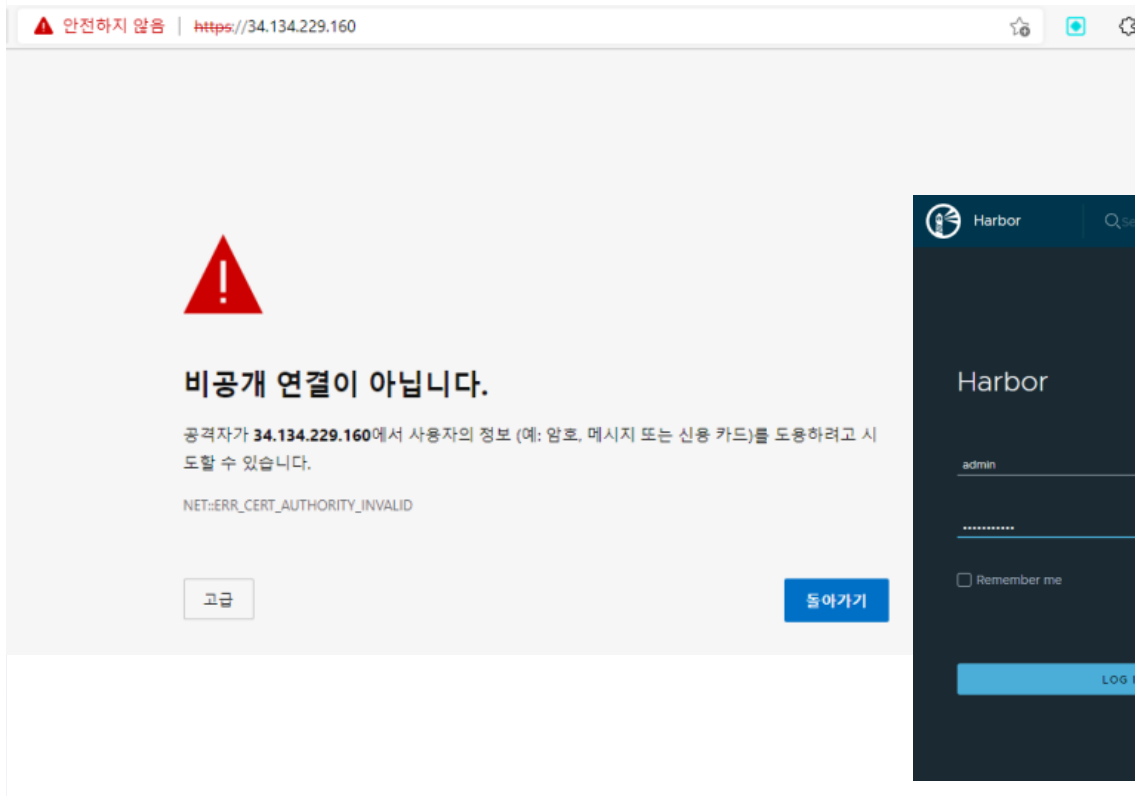
```
./prepare  
./install.sh
```



# harbor를 활용한 컨테이너 레지스트리 구축

## 웹 Harbor 접속

- 웹브라우저로 인스턴스 IP로 접속
- 80포트는 443포트로 리다이렉션
- 고급을 누르고 안전하지 않음을 선택
- Harbor의 초기 ID와 패스워드는 admin//Harbor12345다. 접속을 수행한다.



# harbor를 활용한 컨테이너 레지스트리 구축

## 웹 Harbor 패스워드 변경

- Admin 계정의 패스워드를 변경

The screenshot shows the Harbor web interface. The top navigation bar includes the Harbor logo, a search bar, language selection (English), and a user profile dropdown for 'admin'. The left sidebar contains navigation links for Projects, Logs, Administration, Users, Robot Accounts, Registries, Replications, Distributions, Labels, Project Quotas, Interrogation Services, Garbage Collection, Configuration, and a LIGHT theme toggle. The main content area is titled 'Projects' and displays summary statistics for Projects and Repositories. Below the statistics are buttons for '+ NEW PROJECT' and 'X DELETE'. A table lists the existing projects, with one project named 'library' shown. The user profile dropdown menu is open, highlighting the 'Change Password' option.

Harbor

Search Harbor...

English

admin

Projects

PROJECTS 0 PRIVATE 1 PUBLIC 1 TOTAL

REPOSITORIES 0 PRIVATE 0 PUBLIC 0 TOTAL

+ NEW PROJECT X DELETE

All Projects

<input type="checkbox"/>	Project Name	Access Level	Role	Type	Repositories Count	Creation Time
<input type="checkbox"/>	library	Public	Project Admin	Project		10/26/21, 1:18 AM

Page size 15 1 - 1 of 1 items

User Profile

Change Password

About

Log Out

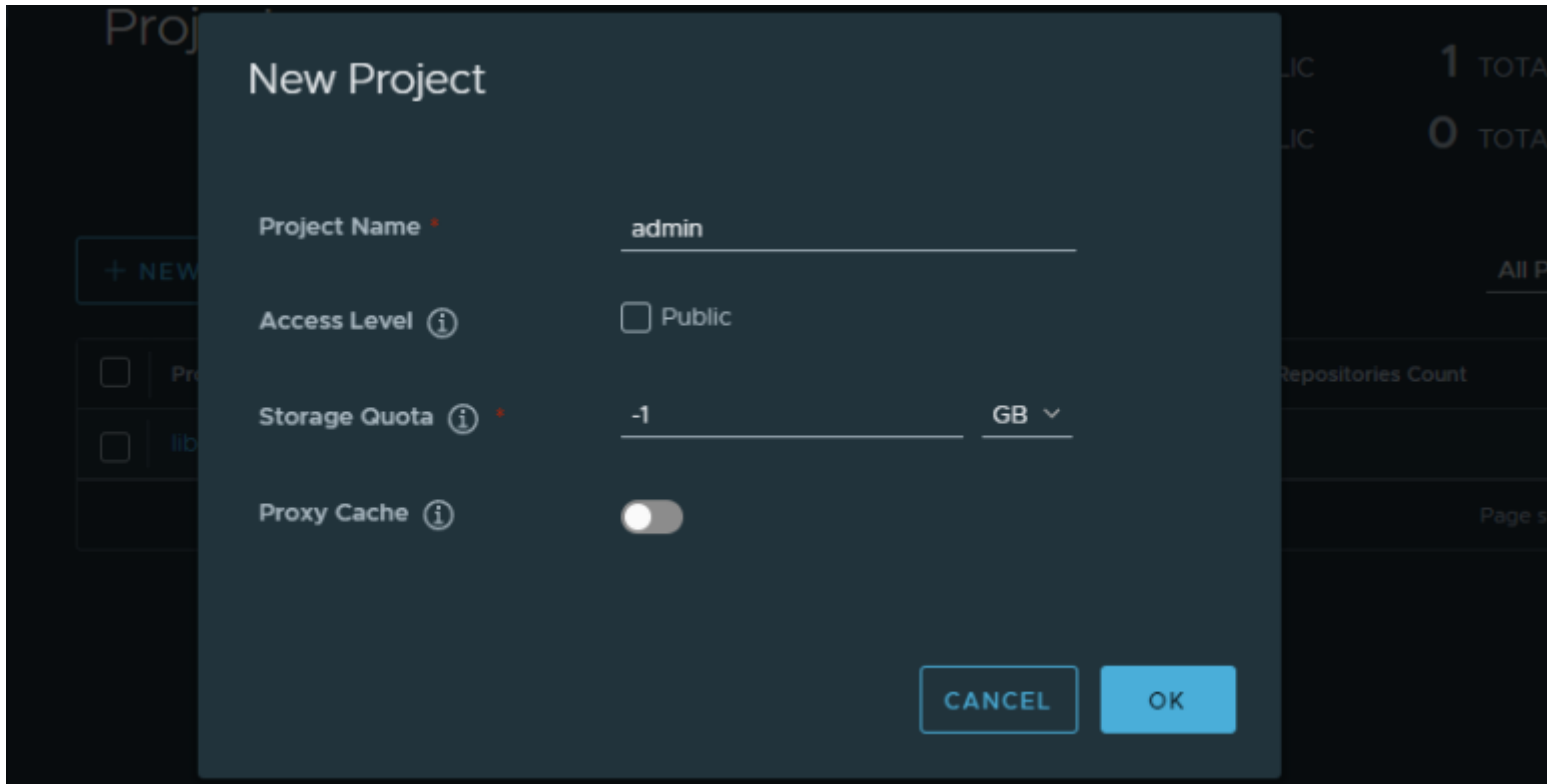
LIGHT

Harbor API V2.0

# harbor를 활용한 컨테이너 레지스트리 구축

## 웹 Harbor 프로젝트 생성

- 처음 로그인하면 기본 프로젝트인 Library가 보임
- 이 프로젝트는 누구나 액세스해서 사용할 수 있는 Public 모드로 구성
- 여기에 New Project 버튼을 눌러 새 프로젝트를 생성



The screenshot shows a 'New Project' modal window in the Harbor web interface. The modal has a dark blue background with white text. It contains the following fields and controls:

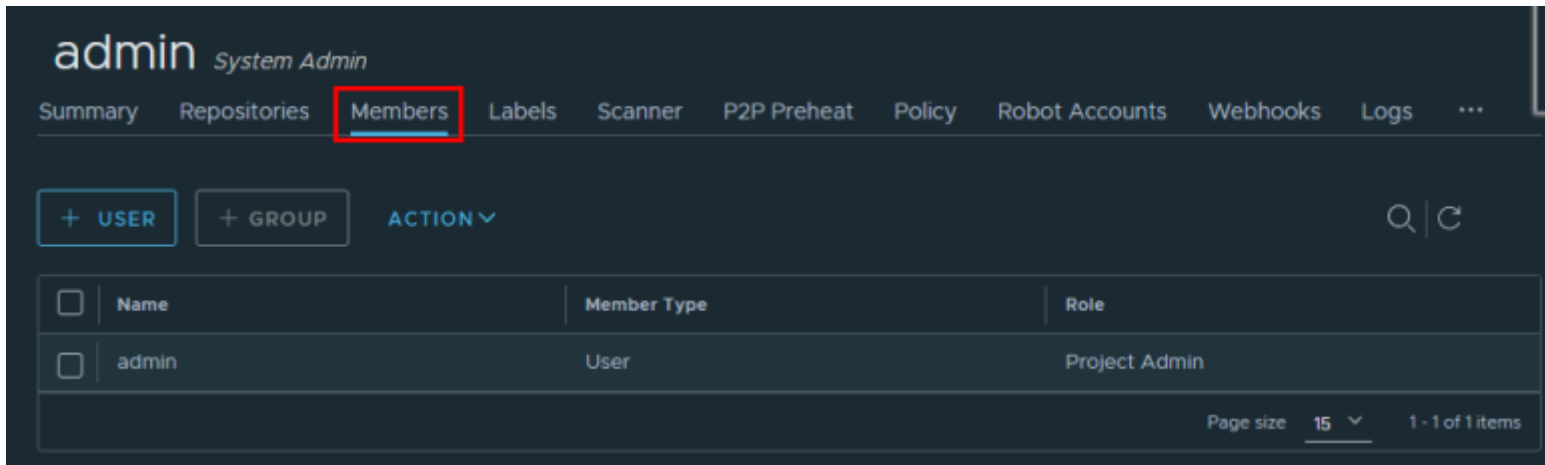
- Project Name \***: A text input field containing the value 'admin'.
- Access Level (i)**: A checkbox labeled 'Public' which is currently unchecked.
- Storage Quota (i) \***: A text input field containing '-1' and a unit dropdown menu set to 'GB'.
- Proxy Cache (i)**: A toggle switch which is currently turned off.

At the bottom right of the modal, there are two buttons: 'CANCEL' and 'OK'.

# harbor를 활용한 컨테이너 레지스트리 구축

## 프로젝트 멤버 설정

- admin 프로젝트로 가면 해당 기능을 사용할 수 있는 유저를 정할 수 있는 기능
- 프로젝트 단위대로 권한을 부여해 액세스할 수 있는 유저 구성
- 유저는 왼쪽 유저 탭에서 새로운 유저를 구성 가능



# harbor를 활용한 컨테이너 레지스트리 구축

## ▶ docker CLI를 활용한 Harbor 사용

- 도커 CLI에서 하버로 접속하는 설정
- admin 유저를 사용해 원하는 이미지를 업로드하기 위해 로그인

```
docker login 34.134.229.160 -u admin -p Harbor12345
```

- admin 권한으로 업로드를 진행

- nginx를 pulling하고 nginx에 태그를 추가
- 전달되는 IP는 Harbor의 IP, 그리고 추가된 태그로 푸시를 진행

```
docker pull nginx  
docker tag nginx 34.134.229.160/admin/nginx  
docker push 34.134.229.160/admin/nginx
```