

CI/CD

By: Puskar Adhikari

Introduction

- **CI (Continuous Integration)**

Developers frequently merge code into a shared branch (like **main**). Each merge triggers **automated tests**, linting, and builds to catch errors early.

- **CD (Continuous Deployment / Delivery)**

After CI passes, code is automatically deployed to a **staging or production** server. This reduces manual deployment effort and ensures consistent delivery.

The Flow

Example flow:

Developer → Pushes code → Gitlab CI runs tests → Docker builds image → Deploys to server or cloud (AWS, Vercel, Render, etc.)

Gitlab CI

- GitLab automatically runs pipelines using the file `.gitlab-ci.yml` in the repository root.

Stages:

- **install**
- **test**
- **build**
- **deploy**

Gitlab CI











- Go to Settings/
- Go to CI/CD
- Add the variables, your docker username and password (access token)

alt-fsad-2025 / LabReference / FullStackApplication / CI/CD Settings

[Save changes](#)

Project variables

Variables can be accidentally exposed in a job log, or maliciously sent to a third party server. The masked variable feature can help reduce the risk of accidentally exposing variable values, but is not a guaranteed method to prevent malicious users from accessing variables. [How can I make my variables more secure?](#)

CI/CD Variables </> 2	Reveal values	Add variable	
Key ↑	Value	Environments	Actions
DOCKERHUB_PASSWORD  Masked Expanded 	All (default) 	 
DOCKERHUB_USERNAME  Expanded 	All (default) 	 

Group variables (inherited)

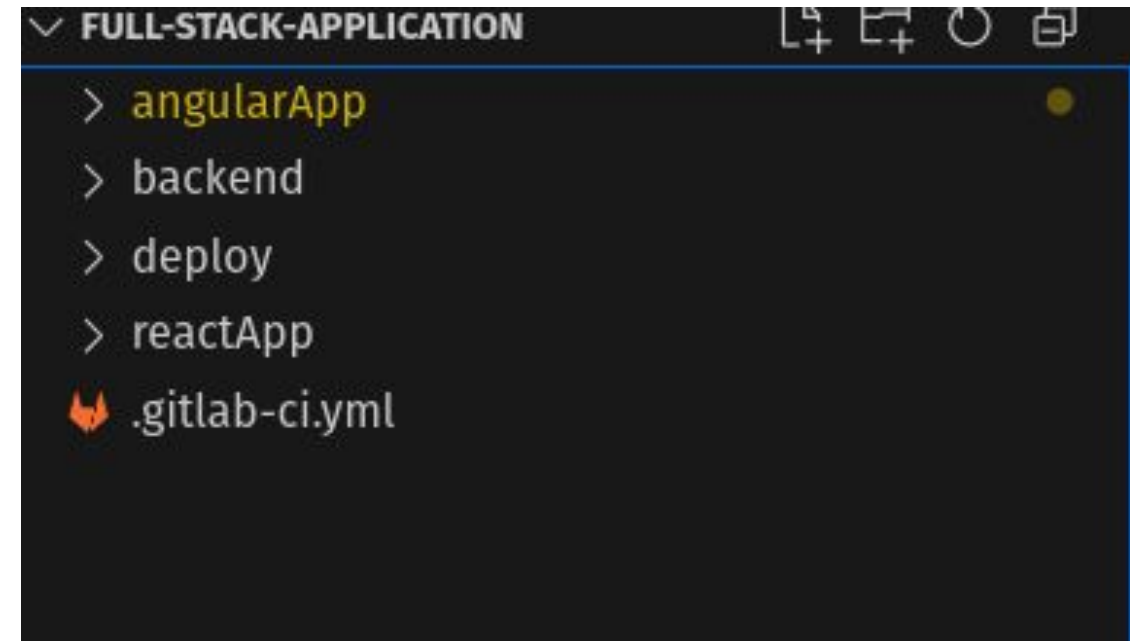
These variables are inherited from the parent group.

CI/CD Variables </> 0

Gitlab CI

- At the root of your project, make a .gitlab-ci.yml file

- ❑ stages:
 - ❑ - build
- ❑ include:
 - ❑ - local: 'backend/.gitlab-ci.yml'
 - ❑ - local: 'reactApp/.gitlab-ci.yml'



Gitlab CI

- Go to backend/
- Make a .gitlab-ci file there
- Do the same thing inside
reactApp/ but change
accordingly for the reactApp.

stages:

- build

variables:

API_IMAGE: puskr99/my-api

DOCKER_TLS_CERTDIR: ""

api_build:

stage: build

image: docker:dind

services:

- docker:dind

before_script:

- echo "Building api"

- cd backend

- docker login -u "\$DOCKERHUB_USERNAME" -p

"\$DOCKERHUB_PASSWORD"

script:

- docker build -t \$API_IMAGE:latest .

Flow

- Make ci file in all directories: frontend and backend
- ci file will build and push the image to your docker hub repo
- You will have a deploy docker-compose file in the server that will pull the images from your docker hub and run them.

docker-compose.yml

services:

 mongodb:

 image: mongo:6

 restart: always

 ports:

 - "27017:27017"

 volumes:

 - mongo_data:/data/db

 backend:

 image: puskr99/my-api:latest

 restart: always

 depends_on:

 - mongodb

 environment:

 - NODE_ENV=production

 - MONGO_URI=mongodb://mongodb:27017/fullstack_demo

 - PORT=3000

The Flow

- Implement ci in backend, angularApp and reactApp.
- The ci files will make backend docker image, react docker image and angular docker image.
- It will then push these images to your docker hub.
- In ur server, create a docker compose file
which will run all these images along with mongodb, the docker compose will fetch theses images from ur personal docker hub.