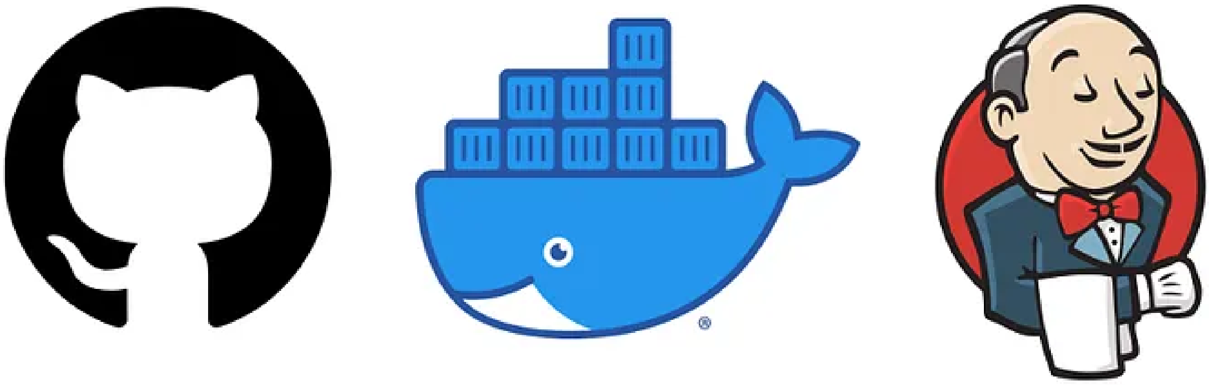
# Setup CI/CD with GitHub, Docker, and Jenkins.

Samer Alsaydali · Follow

6 min read · Sep 17, 2023



In this tutorial, we are going to build and run a simple Node.js app on our (local or remote) server using docker, triggering the build from a GitHub repository git push event, using Jenkins.

### Setup Docker:

**Install Docker Engine**

Choose the best method for you to install Docker Engine. This client-server application is available on Linux, Mac…

docs.docker.com

### Setup Jenkins:

**Installing Jenkins**

Jenkins - an open source automation server which enables developers around the world to reliably build, test, and…

www.jenkins.io

## Step 1) Node App:

4 Files to be created:

1- server.js

The app simply and express app that prints ‘Hello World — version’

```
'use strict';

const express = require('express');

// Constants
const PORT = 8080;
const HOST = '0.0.0.0';

// App
const app = express();
app.get('/', (req, res) => {
  res.send('Hello World - v1.0');
});

app.listen(PORT, HOST, () => {
  console.log(`Running on http://${HOST}:${PORT}`);
});
```

2- package.json

Simply adds the express library to the app.

```
{
  "name": "docker_web_app",
  "version": "1.0.0",
  "description": "Node.js on Docker",
  "author": "Samer Al Saydali <samsaydali@gmail.com>",
  "main": "server.js",
  "scripts": {
    "start": "node server.js"
  },
  "dependencies": {
    "express": "^4.18.2"
  }
}
```

3- Dockerfile

```
FROM node:18

WORKDIR /usr/src/app

COPY package*.json ./

RUN npm install

COPY . .

EXPOSE 8080
CMD [ "node", "server.js" ]
```

The provided Dockerfile appears to be for building a Node.js application container. Let me explain each part of it:

1. `FROM node:18`: This line specifies the base Docker image to use. In this case, it starts with an official Node.js image tagged with version 18. This image contains a pre-installed Node.js runtime.

2. `WORKDIR /usr/src/app`: This line sets the working directory inside the container to `/usr/src/app`. All subsequent commands will be executed in this directory.

3. `COPY package*.json ./`: This line copies the `package.json` and `package-lock.json` (if it exists) from the host machine to the container's current working directory. These files are essential for installing application dependencies using npm.

4. `RUN npm install`: This command installs the Node.js application's dependencies using npm. It reads the `package.json` and `package-lock.json` files and installs all the required packages.

5. `COPY . .`: This line copies the entire content of your application (all files and directories) from the host machine to the current working directory inside the container. This step is used to add your application code to the container.

6. `EXPOSE 8080`: This instruction tells Docker that the container will listen on port 8080. However, it doesn't actually map this port to any specific port on the host. You would need to do that when running the container using the `-p` or `-P` option.

7. `CMD [ "node", "server.js" ]`: This specifies the command to run when the container starts. It runs the Node.js script `server.js`. This is typically the main entry point of your application.

4- .dockerignore

A wise idea to add .dockerignore file with the content:

```
node_modules
npm-debug.log
```

The `.dockerignore` file is used to specify files and directories that should be excluded when building a Docker image. It's similar in concept to `.gitignore`, but it applies to the Docker build context, which is the set of files and directories sent to Docker for building an image.

## Initial Testing

To use this Dockerfile, you would typically build an image from it and then run containers based on that image. Here are the basic Docker commands to build and run the container:

Build the Docker image:

```
docker build -t test-node-app .
```

Run a container based on the image, mapping port 8080 from the container to a port on the host (e.g., 8090 on the host):

```
docker run -p 8090:8080 test-node-app
```

This will start your Node.js application in a container, and you can access it by going to `http://localhost:`8090 in your web browser.

### Step 2) Expose your Jenkins Service:

If you are using a remote server and can access your Jenkins [ip:port], **skip this step**.

If you are using your personal computer, you might want to let Github access your Jenkins service.

**Ngrok** is a tool that allows you to expose a local server running on your machine to the public internet. It creates a secure tunnel from a public endpoint to a port on your local machine, making it accessible from anywhere on the internet. Ngrok is commonly used for various purposes, including web development, testing, and debugging. Here's how you can use ngrok:

Installation:

- You can download and install ngrok from the official website (https://ngrok.com/download).
- After installing it, you may need to authenticate using your Ngrok account to access some advanced features.

Basic Usage:

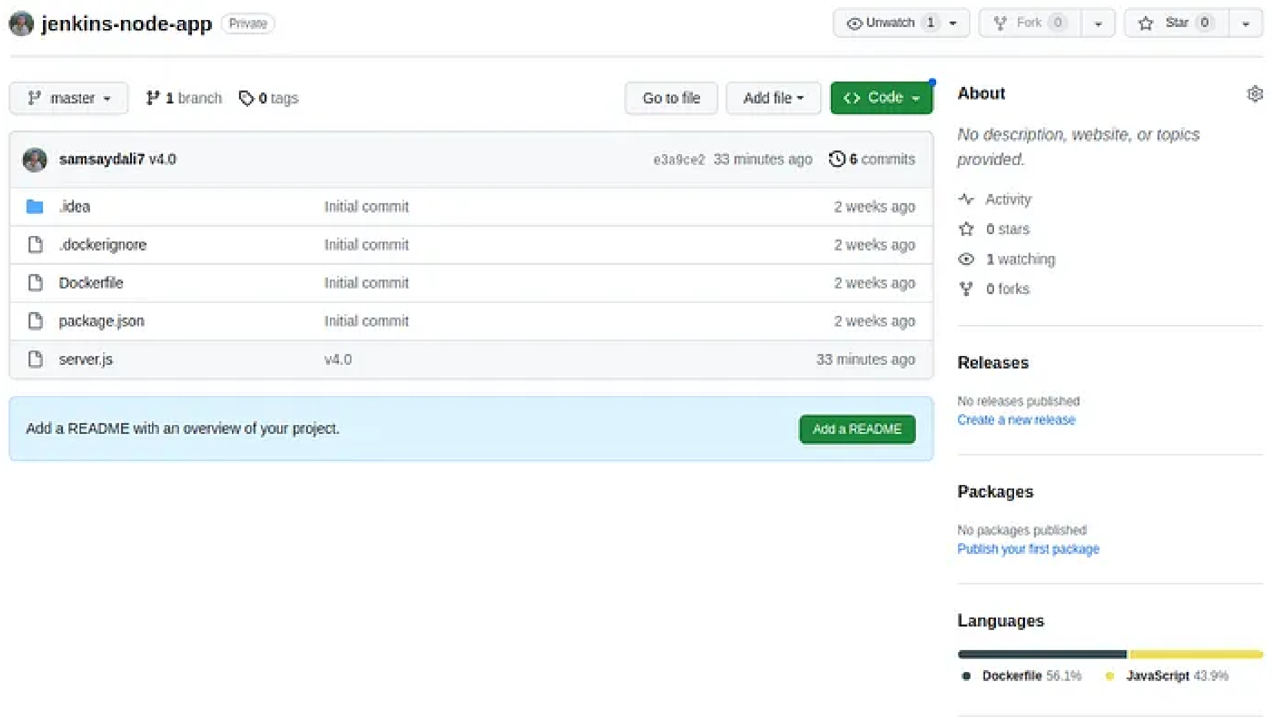
- Open a terminal or command prompt and navigate to the directory where ngrok is installed.
- To expose a local server running on a specific port (e.g., port 8080 for Jenkins), use the following command:

```
ngrok http 8080
```

Ngrok will generate a public URL (usually in the form of `https://randomstring.ngrok.io`) that you can use to access your local server from anywhere.

### Step 3) Setup Github repository:

Create a repository that hosts the previously created files (server.js, package.json, Dockerfile, and .dockerignore).



jenkins-node-app Private

master · 1 branch · 0 tags

samsaydali7 v4.0 — e3a9ce2 33 minutes ago — 6 commits

| | | |
|---|---|---|
| .idea | Initial commit | 2 weeks ago |
| .dockerignore | Initial commit | 2 weeks ago |
| Dockerfile | Initial commit | 2 weeks ago |
| package.json | Initial commit | 2 weeks ago |
| server.js | v4.0 | 33 minutes ago |

Add a README with an overview of your project.

About

No description, website, or topics provided.

Activity · 0 stars · 1 watching · 0 forks

Releases

No releases published
Create a new release

Packages

No packages published
Publish your first package

Languages

Dockerfile 56.1% JavaScript 43.9%

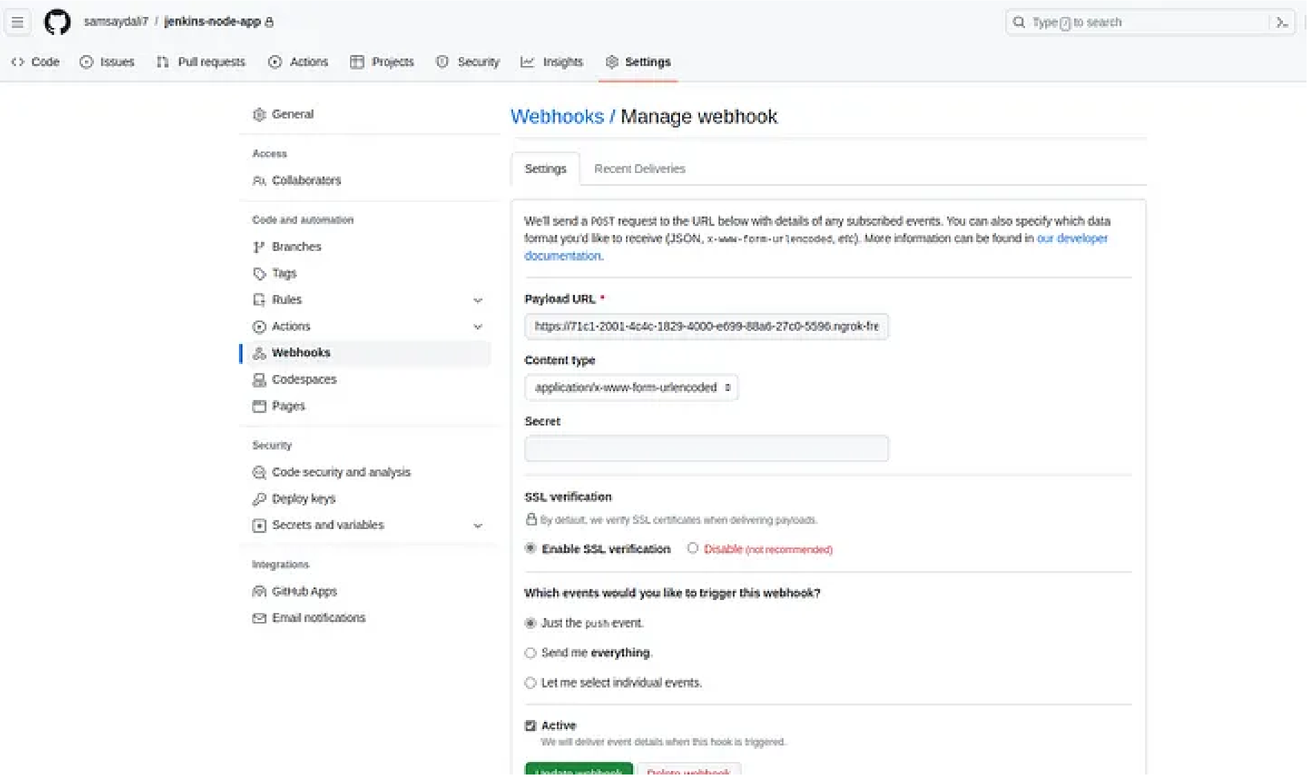
Now add the webhook to jenkins:

Setting up a webhook between GitHub and Jenkins allows you to trigger Jenkins jobs automatically whenever certain events occur in your GitHub repository, such as a code push or a pull request.

In your GitHub repository, go to Settings -> Webhooks -> Add webhook.

Set the Payload URL to `https://your-jenkins-url/github-webhook/`. Replace `your-jenkins-url` with the actual URL of your Jenkins server.

Keep the `/github-webhook/` part.



Webhooks / Manage webhook

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 *
https://71c1-2001-4c4c-1829-4000-e699-88a6-27c0-5596.ngrok-fre

Content type
application/x-www-form-urlencoded

Secret

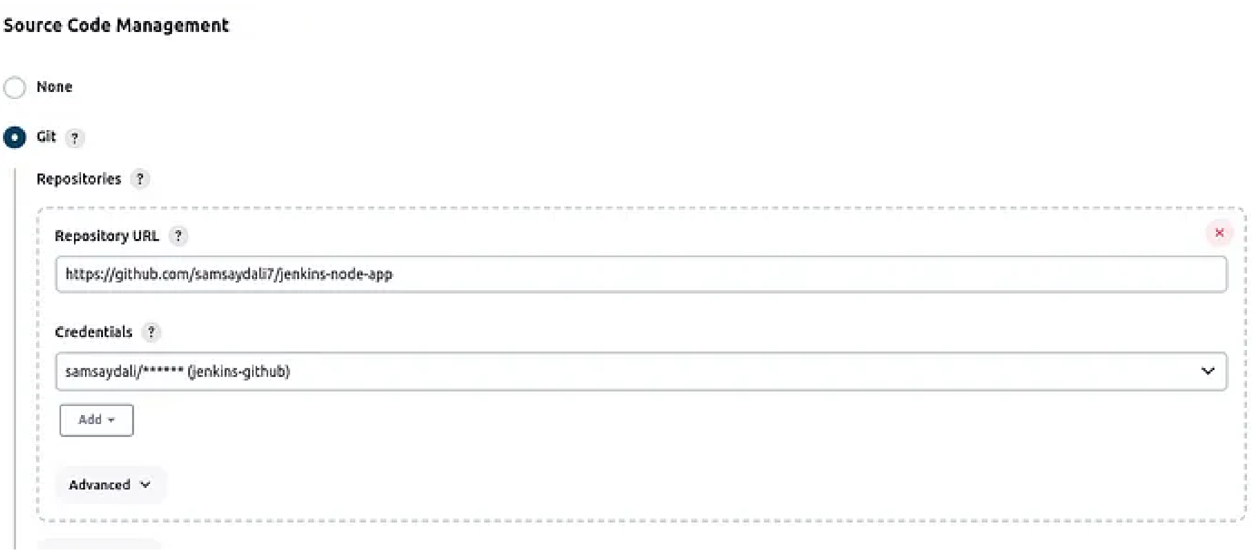
SSL verification
By default, we verify SSL certificates when delivering payloads.
● Enable SSL verification ○ Disable (not recommended)

Which events would you like to trigger this webhook?
● Just the push event.
○ Send me everything.
○ Let me select individual events.

☑ Active
We will deliver event details when this hook is triggered.

## Step 4) Setup Jenkins Project

- Create new freestyle project
- Set the Source Code Management to Git.
- Add your Repository URL



Source Code Management

○ None

● Git ?

Repositories ?

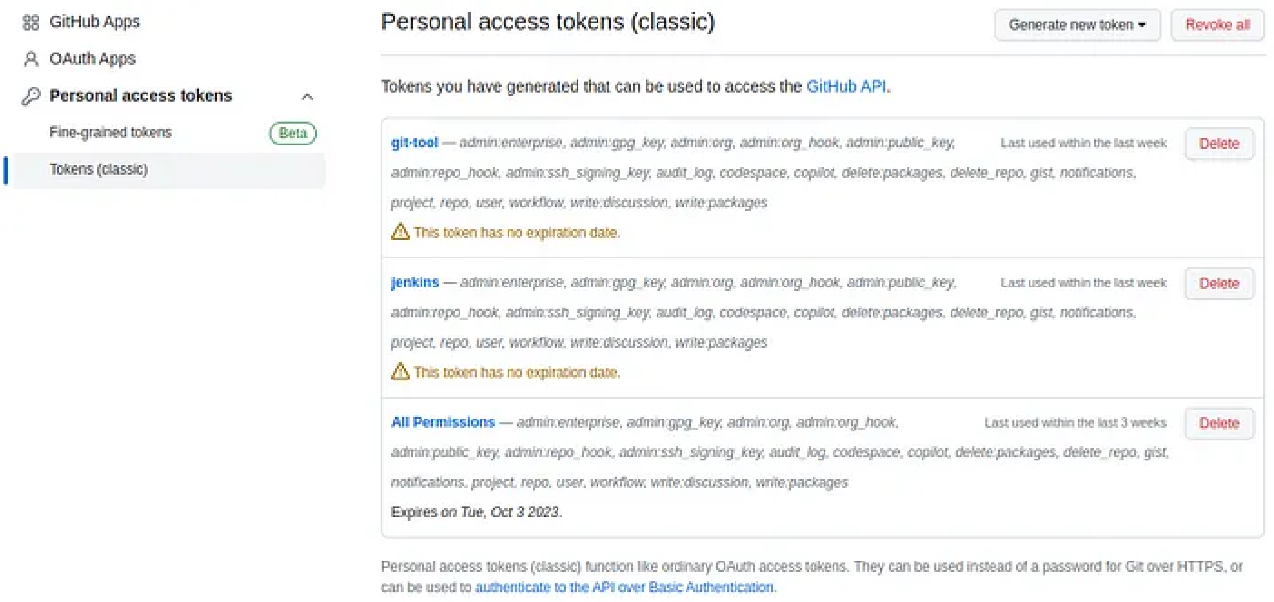
Repository URL ?

https://github.com/samsaydali7/jenkins-node-app

Credentials ?

samsaydali/****** (jenkins-github)

- To securely authenticate Jenkins with your GitHub repository, you should create a GitHub Personal Access Token. Go to your GitHub account settings -> Developer settings -> Personal access tokens -> Generate token. Make sure to give it the necessary permissions, like “repo” for private repositories or specific permissions required for your use case.



Personal access tokens (classic)

Tokens you have generated that can be used to access the GitHub API.

**git-tool** — *admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages* Last used within the last week Delete

⚠ This token has no expiration date.

**jenkins** — *admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages* Last used within the last week Delete

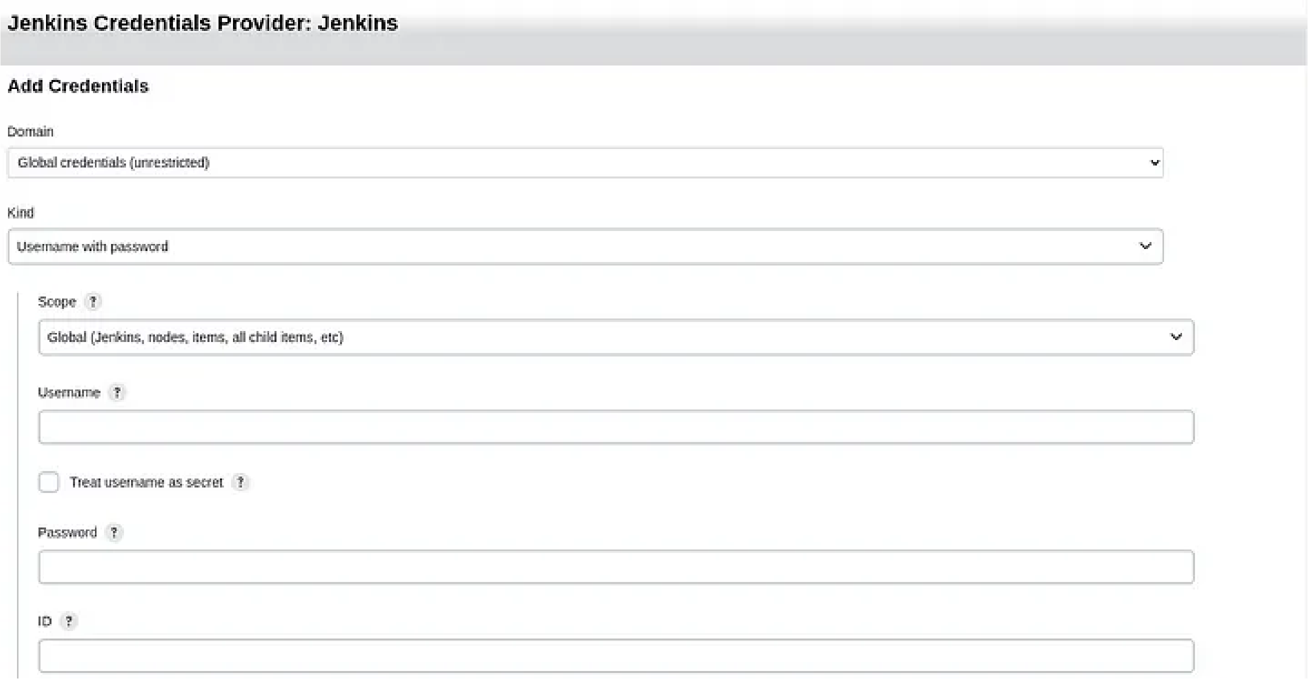
⚠ This token has no expiration date.

**All Permissions** — *admin:enterprise, admin:gpg_key, admin:org, admin:org_hook, admin:public_key, admin:repo_hook, admin:ssh_signing_key, audit_log, codespace, copilot, delete:packages, delete_repo, gist, notifications, project, repo, user, workflow, write:discussion, write:packages* Last used within the last 3 weeks Delete

Expires on *Tue, Oct 3 2023*.

Personal access tokens (classic) function like ordinary OAuth access tokens. They can be used instead of a password for Git over HTTPS, or can be used to authenticate to the API over Basic Authentication.

- For private repositories add and use Jenkins Credentials Provider, set the username to your username, and the password to the Access token created.



Jenkins Credentials Provider: Jenkins

Add Credentials

Domain

Global credentials (unrestricted)

Kind

Username with password

Scope

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

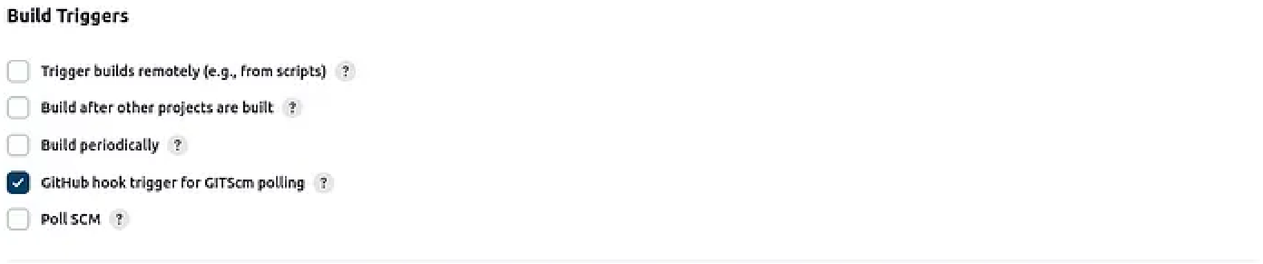
Username

□ Treat username as secret

Password

ID

Set Build Triggers to **GitHub hook trigger for GITScm polling**



Build Triggers

□ Trigger builds remotely (e.g., from scripts)

□ Build after other projects are built

□ Build periodically

☑ GitHub hook trigger for GITScm polling

□ Poll SCM

## Build Steps:

1- Execute Shell

```
#!/bin/bash
sudo docker rmi node-app-img -f
sudo docker build . -t node-app-img
```

Use Docker commands to remove a Docker image and then build a new Docker image from the Dockerfile.

2- Execute Shell

```
#!/bin/bash
PORT_TO_KILL=8080

CONTAINER_ID=$(sudo docker ps -q --filter "expose=$PORT_TO_KILL")

if [ -n "$CONTAINER_ID" ]; then
    sudo docker kill "$CONTAINER_ID"
else
    echo "No container using port $PORT_TO_KILL found."
fi
```

Used to find and kill a Docker container that is using a specific port (`PORT_TO_KILL`).

3- Execute Shell

```
sudo docker run -p 49160:8080 -d node-app-img
```

Runs the Image on port **49160** on your machine.

`-p 49160:8080`: This option is used to map ports between the host and the container. It specifies that port 49160 on the host should be mapped to port 8080 on the container. This means that you can access the application running inside the container on port 8080 by connecting to port **49160** on the host.

### Testing:

Run the build manually on Jenkins then check your http://localhost:49160/, you should see:

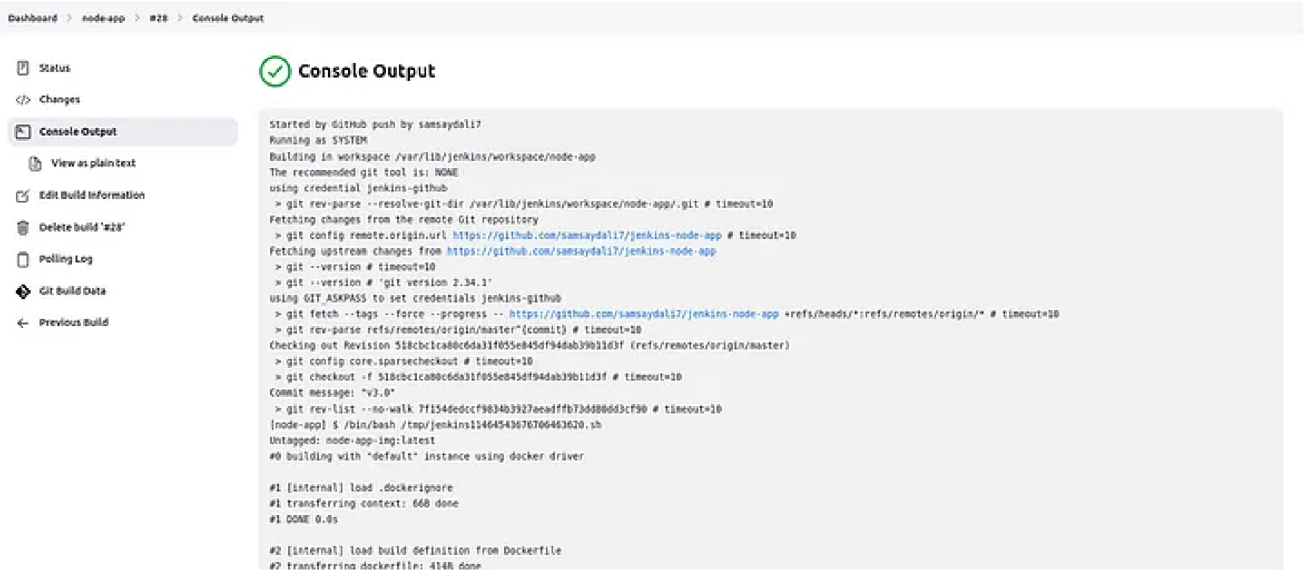
> *Hello World — v1.0*

Now try to make a code change, changing v1.0 to v2.0 for example.

Git add, commit, and push the changes to the remote Github repository.

Check your Jenkins Dashboard:

If you click on your project and view the last run console:



Console Output

```
Started by GitHub push by samsaydali7
Running as SYSTEM
Building in workspace /var/lib/jenkins/workspace/node-app
The recommended git tool is: NONE
using credential jenkins-github
 > git rev-parse --resolve-git-dir /var/lib/jenkins/workspace/node-app/.git # timeout=10
Fetching changes from the remote Git repository
 > git config remote.origin.url https://github.com/samsaydali7/jenkins-node-app # timeout=10
Fetching upstream changes from https://github.com/samsaydali7/jenkins-node-app
 > git --version # timeout=10
 > git --version # 'git version 2.34.1'
using GIT_ASKPASS to set credentials jenkins-github
 > git fetch --tags --force --progress -- https://github.com/samsaydali7/jenkins-node-app +refs/heads/*:refs/remotes/origin/* # timeout=10
 > git rev-parse refs/remotes/origin/master^{commit} # timeout=10
Checking out Revision 518cbc1ca80c6da31f055e845df94dab39b11d3f (refs/remotes/origin/master)
 > git config core.sparsecheckout # timeout=10
 > git checkout -f 518cbc1ca80c6da31f055e845df94dab39b11d3f # timeout=10
Commit message: "v3.0"
 > git rev-list --no-walk 7f154dedccf9834b3927aeadffb73dd80dd3cf90 # timeout=10
[node-app] $ /bin/bash /tmp/jenkins11464543676706463620.sh
Untagged: node-app-img:latest
#0 building with "default" instance using docker driver

#1 [internal] load .dockerignore
#1 transferring context: 66B done
#1 DONE 0.0s

#2 [internal] load build definition from Dockerfile
#2 transferring dockerfile: 414B done
```

Means the the build was triggered and successful, now check your http://localhost:49160/, you should see:

> *Hello World — v2.0*

### Congratulations, you just have created your CI/CD pipeline!

Ci Cd Pipeline · Jenkins · Docker · Github · Webhooks

## Written by Samer Alsaydali

4 Followers · 3 Following

## More from Samer Alsaydali

Samer Alsaydali

## Dockerize a WordPress Website, PhpMyAdmin and its Database.

“Dockerizing” a WordPress website and its database simply means the process of packaging the WordPress application, its dependencies, and…

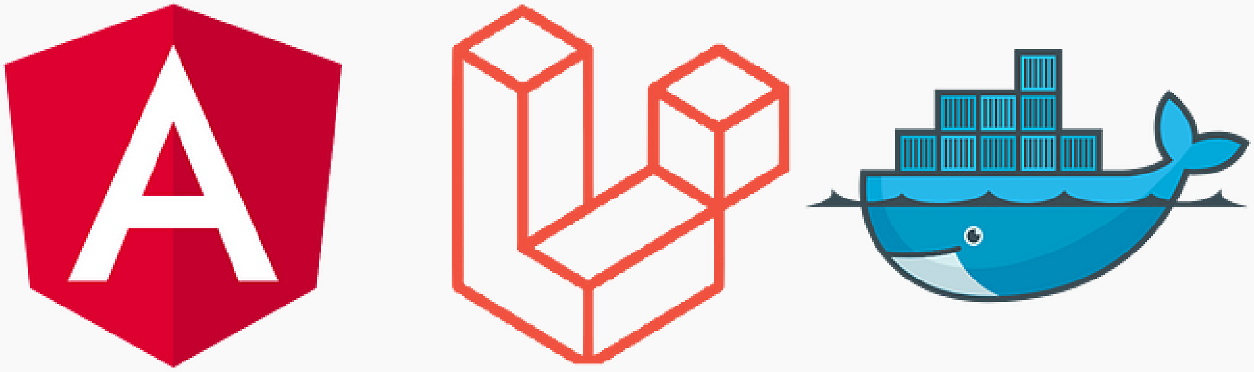
Jan 6 👏 9

Samer Alsaydali

## Create your own Programming Language and its Compiler.

Compiler

May 9, 2023 7 1



Samer Alsaydali

## Run Laravel backend, Angular frontend, and MySQL database with Docker Compose.

In this tutorial article, we are going to setup our already existing full-stack Laravel and Angular application; the backend, the frontend…

Feb 1

Samer Alsaydali

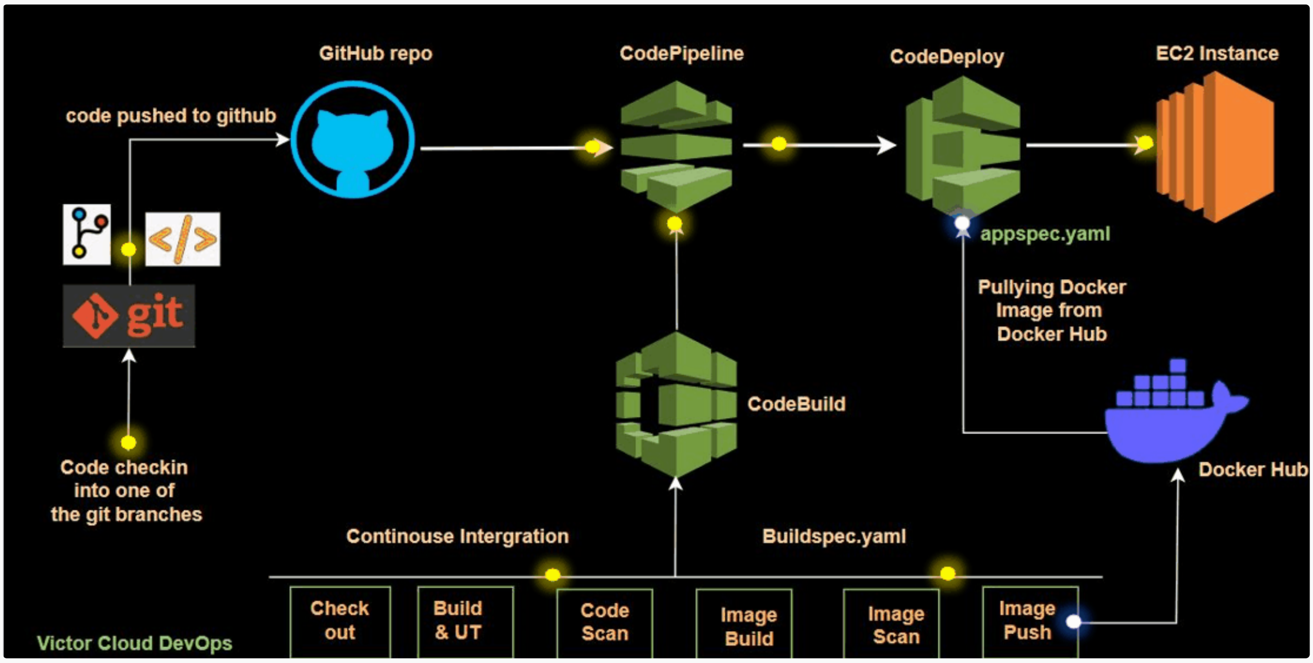
## Code generation for CRUD components based on description files.

Develop a way that automates the generation of CRUD components in your system; developers can get more time to focus on the business logic.

Feb 1, 2023

See all from Samer Alsaydali

## Recommended from Medium



In Admiring Multipotentiality by Victor wasonga onyango

## Automating Continuous Deployment with AWS CodeDeploy and CodePipeline.

In this hands-on guide, we'll dive into setting up Continuous Deployment (CD) using AWS CodeDeploy, a powerful managed service that...

Oct 22 8

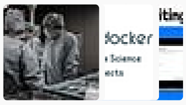
Vinoth Subbiah

## 🔨 Day 11: GitLab CI/CD for Docker-based Workflows

🚀 Learning Objective: Learn how to build, test, and deploy Docker containers in your GitLab CI/CD pipeline. Understand how to set up...

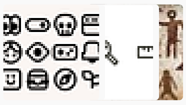
✦ Sep 26 👏 11

## Lists



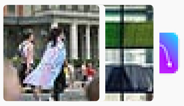
### Coding & Development

11 stories · 933 saves



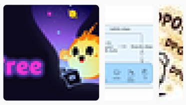
### Icon Design

36 stories · 458 saves



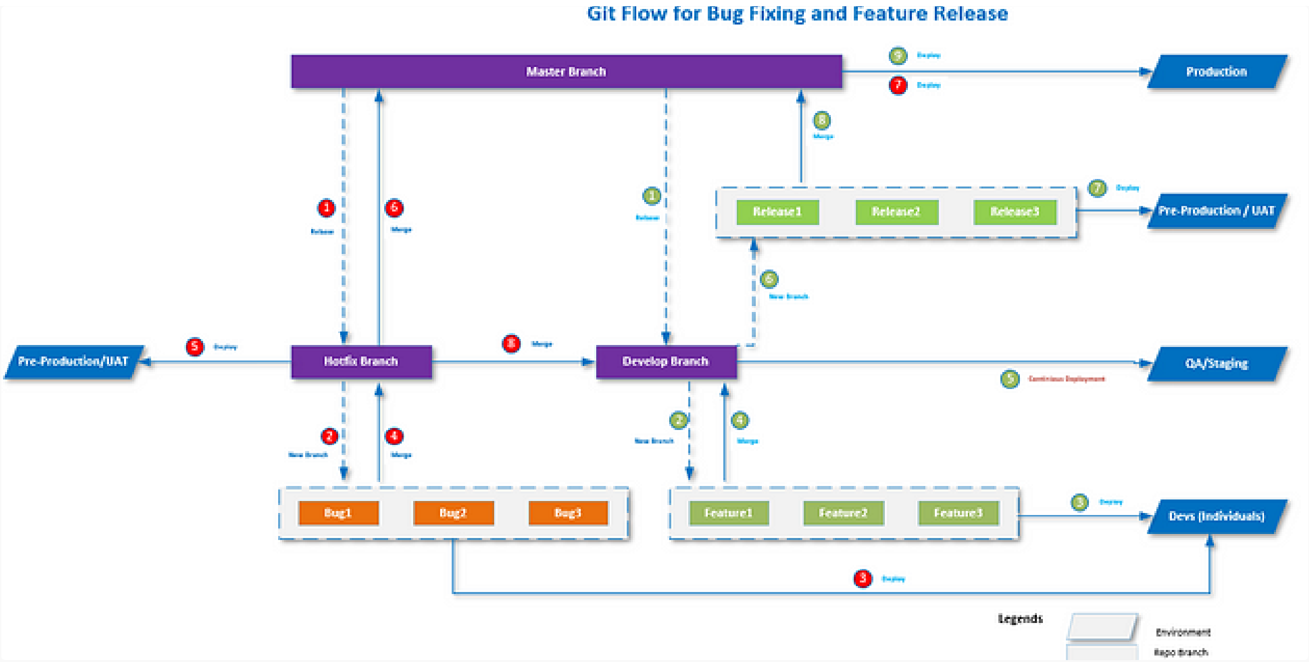
### Generative AI Recommended Reading

52 stories · 1546 saves



### Natural Language Processing

1851 stories · 1473 saves

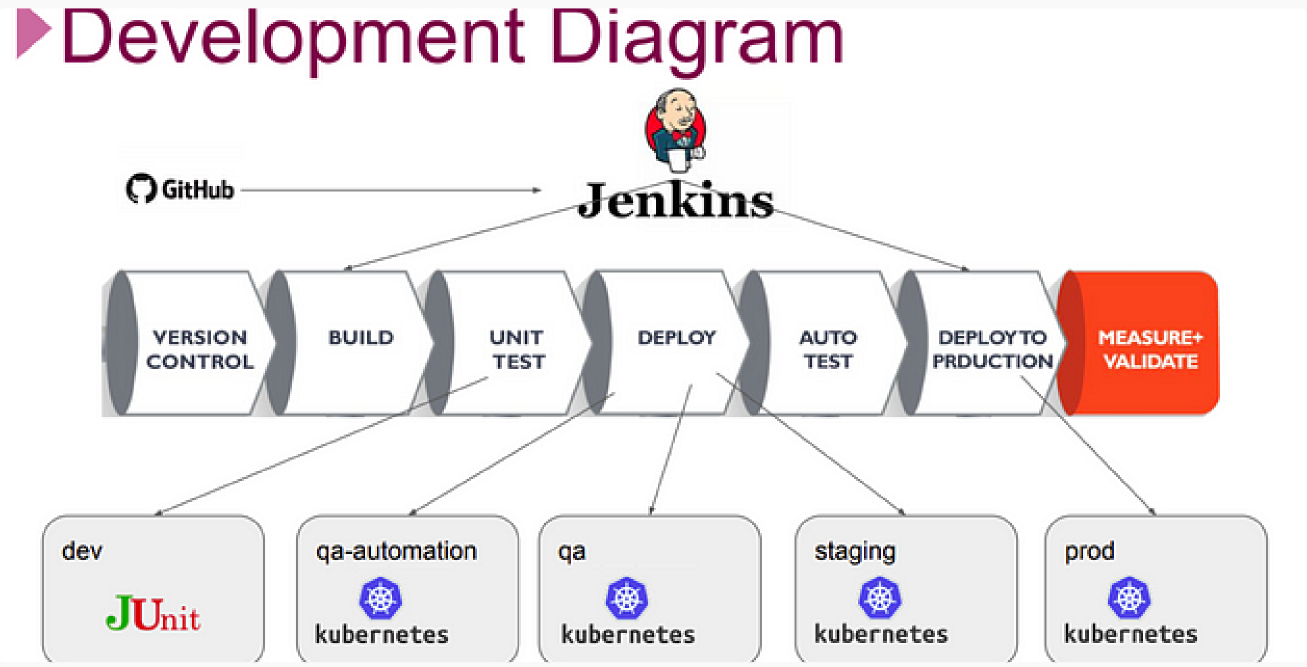


Bibhu Mishra

## Git 3-Branch Strategy

A Git branching strategy is crucial for managing development, bug fixes, and deployment across various environments. The chosen strategy...

Sep 13 2

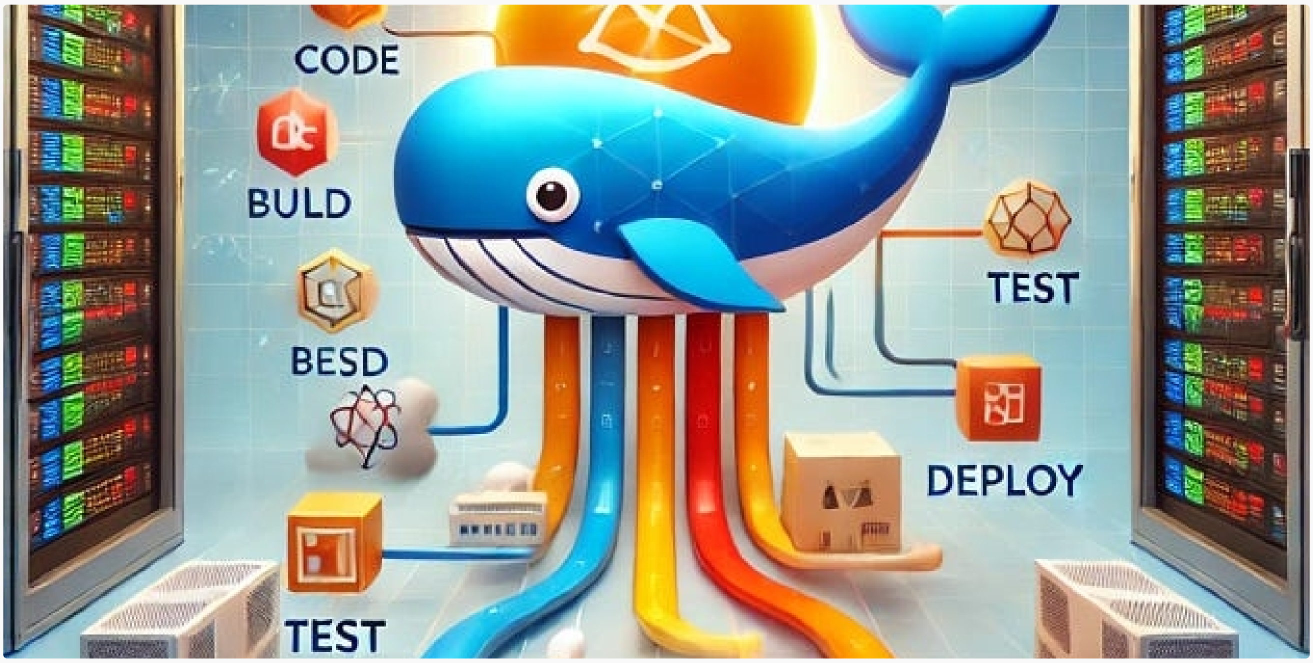


Adnan Turgay Aydin

## Building a Real-Time CI/CD Pipeline: Step-by-Step Guide for Microservices Projects

This DevOps project is designed to enhance your resume and provide hands-on experience with a real-world application.

✦ Aug 16 21

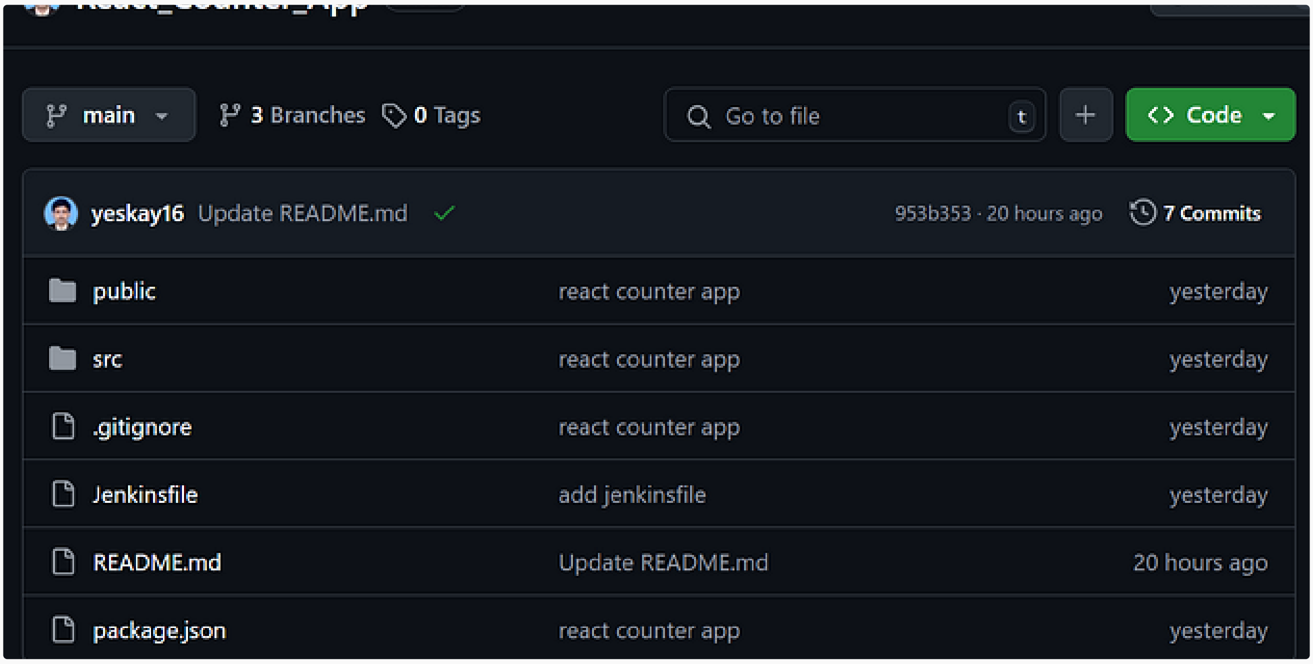


In devsecops-community by Karthick Dkk

## Day 25: How to Automate Docker Builds with Jenkins: Step-by-Step Tutorial

Setting Up Jenkins for Docker Automation: A Quick Guide

✦ Nov 18 7 1



yeskay16 Update README.md ✓ 953b353 · 20 hours ago 7 Commits

| | | |
|---|---|---|
| public | react counter app | yesterday |
| src | react counter app | yesterday |
| .gitignore | react counter app | yesterday |
| Jenkinsfile | add jenkinsfile | yesterday |
| README.md | Update README.md | 20 hours ago |
| package.json | react counter app | yesterday |

Karthik Seenuvasan

**Trigger Jenkins—Multibranch Pipleline using Jenkinsfile.**

https://youtu.be/vQmS0cEjTyw

✦ Sep 25 106

See more recommendations