# Devops-build — Complete step‑by‑step guide

Repository: https://github.com/sriram-R-krishnan/devops-build

This document walks you through **cloning, dockerizing, building, pushing, deploying, and monitoring** the application. It includes all required files (Dockerfile, docker-compose.yml, build.sh, deploy.sh, Jenkinsfile), exact CLI commands, AWS steps, and a checklist of screenshots to capture.

## Table of contents

1. Overview & prerequisites
2. Environment & accounts to prepare
3. Clone the repo
4. Dockerize the app (Dockerfile + .dockerignore)
5. Docker Compose
6. Build and deploy scripts (build.sh, deploy.sh)
7. Git workflow (dev & master) — CLI commands only
8. Docker Hub: dev & prod repos
9. Jenkins: install, configure, pipeline (auto trigger from GitHub)
10. AWS: launch t2.micro, security group, deploy app on EC2
11. Monitoring (Prometheus + Grafana) + Alertmanager or Uptime Kuma (simpler)
12. Screenshots checklist (what to capture & filenames)
13. Troubleshooting & tips
14. Appendix: full file contents (Dockerfile, docker-compose.yml, build.sh, deploy.sh, Jenkinsfile, Prometheus & Alertmanager basics)

## 1. Overview & prerequisites

* OS: Linux for CI/CD host (or Windows WSL2). Commands below are for Linux/macOS; Windows users may adapt for PowerShell.
* Tools (installed locally and on CI server where needed): git, docker, docker-compose, ssh, openssl (optional), aws-cli (optional).
* Accounts:
  + GitHub account with repository access.
  + Docker Hub account (to create dev (public) and prod (private) repositories).
  + Docker Hub credentials stored in Jenkins credentials store.
  + AWS account to launch EC2 t2.micro.
* Ports: app must be accessible on port 80 (HTTP).

## 2. Environment & accounts to prepare

1. Create two Docker Hub repos under your Docker Hub username: yourhub/dev (public), yourhub/prod (private).
2. Create a GitHub personal access token (permissions: repo, admin:repo\_hook) to allow Jenkins to access the repo if using webhooks.
3. Note your public IP (from https://ifconfig.me or curl ifconfig.me) — we’ll use it to restrict SSH access to EC2.

## 3. Clone the repo

# local machine  
git clone https://github.com/sriram-R-krishnan/devops-build.git  
cd devops-build

## 4. Dockerize the app

Create Dockerfile at repo root. Example (adjust if app uses a different language/runtime):

# Use an official Node image if app is node-based (adjust as needed)  
FROM node:18-alpine AS build  
WORKDIR /app  
COPY package\*.json ./  
RUN npm ci --only=production  
COPY . .  
EXPOSE 80  
CMD ["node", "index.js"]

If the app uses Python/Flask or other, replace base image and start command accordingly.

Create .dockerignore to keep images small:

node\_modules  
.git  
.gitignore  
.DS\_Store  
\*.log

Create .gitignore (example):

node\_modules/  
.env  
.DS\_Store

## 5. Docker Compose

Create docker-compose.yml for local testing and for deploying monitoring stack later.

version: '3.8'  
services:  
 app:  
 build: .  
 image: yourhub/dev:latest  
 container\_name: dev-app  
 ports:  
 - "80:80"  
 restart: unless-stopped  
 environment:  
 - NODE\_ENV=production

## 6. Build and deploy scripts

Add executable permissions: chmod +x build.sh deploy.sh.

### build.sh

#!/usr/bin/env bash  
set -e  
# Usage: ./build.sh <branch-or-tag> <image-tag>  
BRANCH=${1:-dev}  
IMAGETAG=${2:-latest}  
HUB\_USER=yourhub  
DEV\_REPO=${HUB\_USER}/dev  
PROD\_REPO=${HUB\_USER}/prod  
  
# Build image  
docker build -t ${DEV\_REPO}:${IMAGETAG} .  
  
echo "Built ${DEV\_REPO}:${IMAGETAG}"  
  
# Login to docker hub (expects DOCKERHUB\_USERNAME & DOCKERHUB\_PASSWORD env vars or use `docker login` before)  
# docker login -u $DOCKERHUB\_USERNAME -p $DOCKERHUB\_PASSWORD  
  
docker push ${DEV\_REPO}:${IMAGETAG}  
  
echo "Pushed to ${DEV\_REPO}:${IMAGETAG}"

### deploy.sh (deploys image to remote server via SSH)

#!/usr/bin/env bash  
set -e  
# Usage: ./deploy.sh <user@server> <image> <container\_name>  
REMOTE=${1}  
IMAGE=${2}  
CNAME=${3:-app}  
PORT=${4:-80}  
  
# Pull and restart on remote host  
ssh -o StrictHostKeyChecking=no ${REMOTE} "docker pull ${IMAGE} && docker stop ${CNAME} || true && docker rm ${CNAME} || true && docker run -d --name ${CNAME} -p ${PORT}:80 --restart unless-stopped ${IMAGE}"  
  
echo "Deployed ${IMAGE} to ${REMOTE} as ${CNAME}"

## 7. Git workflow (CLI commands only)

# create dev branch and push  
git checkout -b dev  
# make changes, add Dockerfile, docker-compose, scripts  
git add Dockerfile docker-compose.yml build.sh deploy.sh .dockerignore .gitignore  
git commit -m "Dockerize app + CI/CD scripts"  
# push to remote dev branch  
git push origin dev

To merge dev->master using CLI:

# on local machine  
git checkout master  
git pull origin master  
git merge --no-ff dev -m "merge dev into master"  
git push origin master

## 8. Docker Hub: dev & prod repos

1. Create repos yourhub/dev (public) and yourhub/prod (private).
2. From CI (Jenkins), authenticate and push to the correct repo depending on branch.

## 9. Jenkins — install, configure, pipeline

### Install Jenkins

On a server (or locally):

# for Ubuntu (example)  
wget -q -O - https://pkg.jenkins.io/debian/jenkins.io.key | sudo apt-key add -  
sudo sh -c 'echo deb https://pkg.jenkins.io/debian-stable binary/ > /etc/apt/sources.list.d/jenkins.list'  
sudo apt update  
sudo apt install openjdk-11-jdk -y  
sudo apt install jenkins -y  
sudo systemctl start jenkins  
sudo systemctl enable jenkins

Open Jenkins on port 8080 and complete initial setup, install recommended plugins.

### Required plugins

* Git plugin
* Pipeline
* Docker Pipeline
* GitHub Integration / GitHub plugin
* Credentials Binding

### Credentials to add (in Jenkins Credentials store)

* Docker Hub username & password (as ‘dockerhub-creds’)
* GitHub personal access token (if using private repo or to create webhooks)
* SSH key for EC2 server (if Jenkins will deploy to EC2 via SSH)

### Create Pipeline job (multibranch or declarative pipeline)

Use a Jenkinsfile in the repo root. Example Jenkinsfile (declarative):

pipeline {  
 agent any  
 environment {  
 HUB\_USER = 'yourhub'  
 DOCKERHUB\_CRED = credentials('dockerhub-creds')  
 }  
 stages {  
 stage('Checkout') {  
 steps {  
 git url: 'https://github.com/sriram-R-krishnan/devops-build.git', credentialsId: 'github-token'  
 }  
 }  
 stage('Build Image') {  
 steps {  
 script {  
 def imageName = "${HUB\_USER}/${env.BRANCH\_NAME == 'master' ? 'prod' : 'dev'}:${env.BUILD\_NUMBER}"  
 sh "docker build -t ${imageName} ."  
 withCredentials([usernamePassword(credentialsId: 'dockerhub-creds', passwordVariable: 'DH\_PASS', usernameVariable: 'DH\_USER')]) {  
 sh "echo $DH\_PASS | docker login -u $DH\_USER --password-stdin"  
 sh "docker push ${imageName}"  
 }  
 env.PUSHED\_IMAGE = imageName  
 }  
 }  
 }  
 stage('Deploy to Server') {  
 steps {  
 // If you want Jenkins to deploy, configure SSH credentials and run remote docker pull & run  
 echo "Deployment handled via deploy script or Jenkins SSH"  
 }  
 }  
 }  
}

## 10. AWS: Launch t2.micro, SG, deploy app on EC2

### Steps (AWS Console or AWS CLI)

1. Login to AWS console.
2. Navigate to EC2 -> Launch Instance.
   * AMI: Amazon Linux 2 or Ubuntu 22.04 LTS.
   * Instance type: t2.micro
   * Key pair: create/download a key pair (keep .pem file safe). This key will be used to SSH.
3. Security Group (important):
   * HTTP 80: Source 0.0.0.0/0 (anyone can access the app via browser)
   * SSH 22: Source <your-public-ip>/32 **(restrict to your IP only)**
4. Launch instance.

#### Using AWS CLI to open SSH to only your IP:

MY\_IP=$(curl -s ifconfig.me)  
aws ec2 authorize-security-group-ingress --group-id <sg-id> --protocol tcp --port 22 --cidr ${MY\_IP}/32  
aws ec2 authorize-security-group-ingress --group-id <sg-id> --protocol tcp --port 80 --cidr 0.0.0.0/0

### Deploy app on EC2 (assume instance has Docker installed)

SSH into server:

ssh -i ~/keys/mykey.pem ec2-user@<ec2-public-ip>  
# install docker (Amazon Linux 2 example)  
sudo yum update -y  
sudo amazon-linux-extras install docker -y  
sudo service docker start  
sudo usermod -a -G docker $USER  
# either logout/login or use sudo docker

Pull and run the image pushed to Docker Hub:

docker pull yourhub/prod:latest  
docker run -d --name app -p 80:80 --restart unless-stopped yourhub/prod:latest

Or use deploy.sh from your CI server to SSH & run these commands automatically.

## 11. Monitoring

### Prometheus + Grafana + Alertmanager

**Prometheus** can scrape a metrics endpoint. If your app does not expose Prometheus metrics, use blackbox\_exporter to probe the HTTP endpoint or node\_exporter + cAdvisor for container metrics.

Minimal Prometheus prometheus.yml example to scrape blackbox exporter:

scrape\_configs:  
 - job\_name: 'blackbox'  
 metrics\_path: /probe  
 params:  
 module: [http\_2xx]  
 static\_configs:  
 - targets: ['http://<ec2-public-ip>/']  
 relabel\_configs:  
 - source\_labels: [\_\_address\_\_]  
 target\_label: \_\_param\_target  
 - source\_labels: [\_\_param\_target]  
 target\_label: instance  
 - target\_label: \_\_address\_\_  
 replacement: blackbox:9115 # blackbox exporter

Run Prometheus + blackbox via docker-compose and add Alertmanager config to send email on probe\_failed alerts.