# **How to Dockerize NodeJS app**

# **1. Get your nodejs app**

# **2. Setup Docker — Build NodeJS app Docker Image**

Now that we have our nodeJS app. Let’s set up the docker build context to build the docker image.

*****Create dockerignore*****

We’ll start by creating the .dockerignore in the root directory of our project. The ****dockerignore**** file is similar to gitignore file, used by the git tool. Similar to .gitignore file, it allows you to specify a pattern for files and folders that should be ****ignored**** by the ****Docker**** client when generating a build context.  .dockerignore file looks like this:

# ----------------------------------------------------------------\*

# This will prevent your local modules and debug logs from being copied onto your

# Docker image and possibly overwriting modules installed within your image.

# ----------------------------------------------------------------\*

node\_modules

npm-debug.log

# ignore .git and .cache folders

.git

.cache

# ignore all markdown files (md) beside all README\*.md other than README-secret.md

\*.md

!README\*.md

README-secret.md

# github related files

.github/

*****Create Dockerfile*****

Dockerfile is like a blueprint that defines how the docker image will be built and specifies what will be included in our application when the image is running in a docker container. Dockerfile file looks like this:

FROM node

LABEL authors="Yann Mulonda"

# update dependencies and install curl

RUN apt-get update && apt-get install -y \

curl \

&& rm -rf /var/lib/apt/lists/\*

# Create app directory

WORKDIR /app

# Install app dependencies

# A wildcard is used to ensure both package.json AND package-lock.json are copied

# where available (npm@5+)

# COPY package\*.json ./ \

# ./source ./

# This will copy everything from the source path

# --more of a convenience when testing locally.

COPY . .

# update each dependency in package.json to the latest version

RUN npm install -g npm-check-updates \

ncu -u \

npm install \

npm install express \

npm install babel-cli \

npm install babel-preset \

npm install babel-preset-env

# If you are building your code for production

RUN npm ci --only=production

# Bundle app source

COPY . /app

EXPOSE 3000

CMD [ "babel-node", "app.js" ]

*****Build app docker image*****

Now let’s build the application image using the docker build command. The -t flag with docker build will allow you to tag the image with a memorable name:

docker build . -t your\_dockerhub\_username/nodejs-demo

Make sure you replace your\_dockerhub\_username with your own Docker Hub username and name your app image with the name of your choice. This is important because we are going to push the image to Docker Hub. Once the docker image build is complete, check your images:

docker images

*****Run the app image in a container*****

Let’s create a container with the app docker image built previously using [docker run](https://docs.docker.com/engine/reference/commandline/run/" \t "https://blog.devgenius.io/_blank):

docker run --name nodejs-app-demo -it -d -p 80:80 yanndocker/nodejs-demo

* -p: This publishes the port on the container and maps it to a port on our host. Change to another port if port 80 is not available on your system.
* -d: This runs the container in the background.
* -itstarts the container in the interactive mode that allows you to interact with /bin/bash of the container.
* --name: give the container a meaningful name.

Once your container is up and running, you can inspect a list of your running containers with [docker ps](https://docs.docker.com/engine/reference/commandline/ps/" \t "https://blog.devgenius.io/_blank):

docker ps

# **3. GitHub Action CI/CD & push Image to Docker Hub**

We’ll now set CI/CD to our project and push the image to Docker Hub.

*****Setup Github Actions CI/CD*****

*****Importnat Note — before you push the new defined Github workflow to your remote git repo, make sure that you have done the following:*****

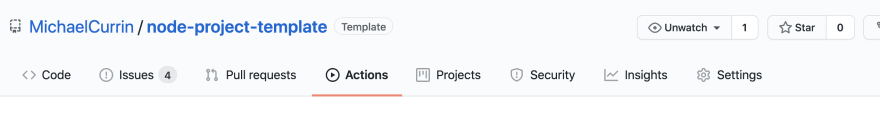
* Added your Docker username and token on your GitHub repo as secrets
* Create your project on Docker Hub where you are going to push the image.

### 4. Choose your repo

Create a new GitHub repo or choose an existing one. It doesn't matter what code is in the repo.

### 5. Create a workflow

Click the "Actions" tab on your repo. Note that you can view the tab on any repo but you can only edit workflows if you have edit access to the repo.

[](https://github.com/MichaelCurrin/node-project-template/actions)

If you have no workflows (config files used for pipelines) yet, you'll be prompted to create one.

Rather than picking a template, choose the ****skip**** option (Skip this and set up a workflow yourself). You can always come back later to add another workflow using a Node.js or Python template for example.

You'll be taken to an editor view for editing a new file called .github/workflows/main.yml. You can leave the path as is.

name: CI

on: push

jobs:

build:

runs-on: ubuntu-latest

steps:

- uses: actions/checkout@v2

- name: Run a one-line script

run: echo "Hello, world!"

The key parts to note are the triggers (on a push event) and the steps (commands to run within a labeled "job"). You can name the job anything (no spaces). You only need one job in most cases.

You can have as many steps as you want. In the last step, the optional name field is just for logging purposed but run is the command that that gets run in the shell. We're using Ubuntu here as that is the typical choice for Actions.

Now ****save**** your file - click "Start commit" in the top right of GH or commit and push locally.

Note that workflow config file uses ****YAML syntax****, which is like JSON

### 6. View logs

You've added a new commit with that file, so your workflow's "push" condition will be triggered and it will run against the current code immediately.

View the Actions tab and find the logs for a run for this workflow - in this case only one run.

Now you have a fully CI/CD pipeline setup for your NodeJS app project that’ll do the following on every code commit and push to the repo:

* build and test your nodejs app
* build the app docker image, test it by running it a container and push that image to your Docker Hub repo for future use, app sharing, and deployment to other platforms.