Michael Herman

Dockerizing an Angular App

Last update: May 20, 2019 • docker, angular

Docker is a containerization tool used to streamline application development and deployment workflows across various environments.

This tutorial shows how to Dockerize an Angular app, built with the Angular CLI, using Docker along with Docker Compose and Docker Machine for both development and production. We'll specifically focus on-

1. Setting up an image for development with code hot-reloading that includes an instance of Chrome for Karma and Protractor testing
2. Configuring a lean, production-ready image using multistage builds

Updates:

- May 2019:
  - Updated to the latest versions of Docker, Node, Angular, and Nginx.
  - Added explanations for various Docker commands and flags.
  - Added a number of notes based on reader comments and feedback.
  - Fixed the running of Protractor e2e tests.

We will be using:

- Node v12.2.0

Contents

- Project Setup
- Docker
- Docker Machine
- Production

Project Setup

Install the Angular CLI globally:

```
$ npm install -g @angular/cli@7.3.9
```

Generate a new app:

```
$ ng new example
$ cd example
```

Docker

Add a *Dockerfile* to the project root:

```
# base image
FROM node:12.2.0

# install chrome for protractor tests
# set working directory
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install and cache app dependencies
COPY package.json /app/package.json
RUN npm install
RUN npm install -g @angular/cli@7.3.9

# add app
COPY . /app

# start app
CMD ng serve --host 0.0.0.0
```

Add a *.dockerignore* as well:

```
node_modules
.git
.gitignore
```

This will speed up the Docker build process as our local dependencies and git repo will not be sent to the Docker daemon.

Build and tag the Docker image:

```
$ docker build -t example:dev .
```

> ```
> RUN npm install -g @angular/cli@7.3.9 --unsafe
> ```
>
> *Review this issue for more info.*

Then, spin up the container once the build is done:

```
$ docker run -v ${PWD}:/app -v /app/node_modules -p 4201:4200 --rm example:dev
```

What's happening here?

1. The docker run command creates a new container instance, from the image we just created, and runs it.
2. `-v ${PWD}:/app` mounts the code into the container at "/app".

    > `{PWD}` *may not work on Windows. See this Stack Overflow question for more info.*

3. Since we want to use the container version of the "node_modules" folder, we configured another volume: `-v /app/node_modules`. You should now be able to remove the local "node_modules" flavor.
4. `-p 4201:4200` exposes port 4200 to other Docker containers on the same network (for inter-container communication) and port 4201 to the host.

    > *For more, review this Stack Overflow question.*

5. Finally, `--rm` removes the container and volumes after the container exits.

Open your browser to http://localhost:4201 and you should see the app. Try making a change to the `AppComponent`'s template (*src/app/app.component.html*) within your code editor. You should see the app hot-reload. Kill the server once done.

> ```
> $ docker run -it -v ${PWD}:/app -v /app/node_modules -p 4201:4200 --rm example:dev
> ```
>
> *Check your understanding and look this up on your own.*

Use the `-d` flag to run the container in the background:

```
$ docker run -d -v ${PWD}:/app -v /app/node_modules -p 4201:4200 --name foo --rm example:dev
```

Once up, update the Karma and Protractor config files to run Chrome in headless mode.

src/karma.conf.js:

```
module.exports = function (config) {
  config.set({
    basePath: '',
    frameworks: ['jasmine', '@angular-devkit/build-angular'],
    plugins: [
      require('karma-jasmine'),
      require('karma-chrome-launcher'),
      require('karma-jasmine-html-reporter'),
      require('karma-coverage-istanbul-reporter'),
      require('@angular-devkit/build-angular/plugins/karma')
    ],
    client: {
      clearContext: false
    },
    coverageIstanbulReporter: {
      dir: require('path').join(__dirname, '../coverage/example'),
      reports: ['html', 'lcovonly', 'text-summary'],
      fixWebpackSourcePaths: true
    },
    reporters: ['progress', 'kjhtml'],
```

```
    autoWatch: true,
    // updated
    browsers: ['ChromeHeadless'],
    // new
    customLaunchers: {
      'ChromeHeadless': {
        base: 'Chrome',
        flags: [
          '--no-sandbox',
          '--headless',
          '--disable-gpu',
          '--remote-debugging-port=9222'
        ]
      }
    },
    singleRun: false,
    restartOnFileChange: true
  });
};
```

e2e/protractor.conf.js:

```
const { SpecReporter } = require('jasmine-spec-reporter');

exports.config = {
  allScriptsTimeout: 11000,
  specs: [
    './src/**/*.e2e-spec.ts'
  ],
  capabilities: {
    'browserName': 'chrome',
    // new
    'chromeOptions': {
      'args': [
      ]
    }
  },
  directConnect: true,
  baseUrl: 'http://localhost:4200/',
  framework: 'jasmine',
  jasmineNodeOpts: {
    showColors: true,
    defaultTimeoutInterval: 30000,
    print: function() {}
  },
  onPrepare() {
    require('ts-node').register({
      project: require('path').join(__dirname, './tsconfig.e2e.json')
    });
    jasmine.getEnv().addReporter(new SpecReporter({ spec: { displayStacktrace: true } }));
  }
};
```

Run the unit and e2e tests:

```
$ docker exec -it foo ng test --watch=false
$ docker exec -it foo ng e2e --port 4202
```

Stop the container once done:

```
$ docker stop foo
```

Want to use Docker Compose? Add a *docker-compose.yml* file to the project root:

```
version: '3.7'
```

```
    container_name: example
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - '.:/app'
      - '/app/node_modules'
    ports:
      - '4201:4200'
```

Take note of the volumes. Without the anonymous volume (`'/app/node_modules'`), the *node_modules* directory would be overwritten by the mounting of the host directory at runtime. In other words, this would happen:

- *Build* - The `node_modules` directory is created in the image.
- *Run* - The current directory is mounted into the container, overwriting the `node_modules` that were installed during the build.

Build the image and fire up the container:

```
$ docker-compose up -d --build
```

Ensure the app is running in the browser and test hot-reloading again. Try both the unit and e2e tests as well:

```
$ docker-compose exec example ng test --watch=false
$ docker-compose exec example ng e2e --port 4202
```

Stop the container before moving on:

> *Windows Users: Having problems getting the volumes to work properly? Review the following resources:*
>
> 1. *Docker on Windows–Mounting Host Directories*
> 2. *Configuring Docker for Windows Shared Drives*
>
> *You also may need to add `COMPOSE_CONVERT_WINDOWS_PATHS=1` to the environment portion of your Docker Compose file. Review the Declare default environment variables in file guide for more info.*

Docker Machine

To get hot-reloading to work with Docker Machine and VirtualBox you'll need to enable a polling mechanism via chokidar (which wraps `fs.watch`, `fs.watchFile`, and `fsevents`).

Create a new Machine:

```
$ docker-machine create -d virtualbox clever
$ docker-machine env clever
$ eval $(docker-machine env clever)
```

Grab the IP address:

```
$ docker-machine ip clever
```

Then, build the images:

```
$ docker build -t example:dev .
$ docker run -it -v ${PWD}:/app -v /app/node_modules -p 4201:4200 --rm example:dev
```

Test the app again in the browser at http://DOCKER_MACHINE_IP:4201 (make sure to replace `DOCKER_MACHINE_IP` with the actual IP address of the Docker Machine). Also, confirm that auto reload is *not* working. You can try with Docker Compose as well, but the result will be the same.

To get hot-reload working, we need to add an environment variable: `CHOKIDAR_USEPOLLING=true`.

```
$ docker run -it -v ${PWD}:/app -v /app/node_modules -p 4201:4200 -e CHOKIDAR_USEPOLLING=true
```

Test it out again. Then, kill the server and add the environment variable to the *docker-compose.yml* file:

```
version: '3.7'

services:

  example:
    container_name: example
    build:
      context: .
      dockerfile: Dockerfile
    volumes:
      - '.:/app'
      - '/app/node_modules'
    ports:
      - '4201:4200'
    environment:
      - CHOKIDAR_USEPOLLING=true
```

Spin up the container. Run the unit tests and e2e tests.

Let’s create a separate Dockerfile for use in production called *Dockerfile-prod*:

```
#############
### build ###
#############

# base image
FROM node:12.2.0 as build

# install chrome for protractor tests
RUN wget -q -O - https://dl-ssl.google.com/linux/linux_signing_key.pub | apt-key add -
RUN sh -c 'echo "deb [arch=amd64] http://dl.google.com/linux/chrome/deb/ stable main" >> /etc
RUN apt-get update && apt-get install -yq google-chrome-stable

# set working directory
WORKDIR /app

# add `/app/node_modules/.bin` to $PATH
ENV PATH /app/node_modules/.bin:$PATH

# install and cache app dependencies
COPY package.json /app/package.json
RUN npm install
RUN npm install -g @angular/cli@7.3.9

# add app
COPY . /app

# run tests
RUN ng test --watch=false
RUN ng e2e --port 4202

# generate build
RUN ng build --output-path=dist
```

```
############

# base image
FROM nginx:1.16.0-alpine

# copy artifact build from the 'build environment'
COPY --from=build /app/dist /usr/share/nginx/html

# expose port 80
EXPOSE 80

# run nginx
CMD ["nginx", "-g", "daemon off;"]
```

Two important things to note:

1. First, we take advantage of the multistage build pattern to create a temporary image used for building the artifact – the production-ready Angular static files – that is then copied over to the production image. The temporary build image is discarded along with the original files, folders, and dependencies associated with the image. This produces a lean, production-ready image.

   In other words, the only thing kept from the first image is the compiled distribution code.

   > *Check out the Builder pattern vs. Multi-stage builds in Docker blog post for more info on multistage builds.*

2. Next, the unit and e2e tests are run in the build process, so the build will fail if the tests do not succeed.

Using the production Dockerfile, build and tag the Docker image:

Spin up the container:

```
$ docker run -it -p 80:80 --rm example:prod
```

Assuming you are still using the same Docker Machine, navigate to http://DOCKER_MACHINE_IP/ in your browser.

Test with a new Docker Compose file as well called *docker-compose-prod.yml*:

```
version: '3.7'

services:

  example-prod:
    container_name: example-prod
    build:
      context: .
      dockerfile: Dockerfile-prod
    ports:
      - '80:80'
```

Fire up the container:

```
$ docker-compose -f docker-compose-prod.yml up -d --build
```

Test it out once more in your browser. Then, break a test in *src/app/app.component.spec.ts* and re-build. It should fail:

```
ERROR: Service 'example-prod' failed to build:
The command '/bin/sh -c ng test --watch=false' returned a non-zero code: 1
```

```
$ eval $(docker-machine env -u)
$ docker-machine rm clever
```

Cheers!

53

What do you think?

74 Responses

👍 Upvote 😝 Funny 😍 Love 😯 Surprised 😤 Angry 😢 Sad

37 Comments **Michael Herman** 1 **Login**

♡ **Recommend** 8 Tweet Share **Sort by Best**

Join the discussion…

LOG IN WITH

OR SIGN UP WITH DISQUS ?

Name

Chris B • 22 days ago

The ng app I am building is using several variables from environment.ts (and environment.prod.ts in prod) and it appears these variables are not being read/used when "ng serve" is being run. The app errs out with a TypeError: Cannot read property 'foo' of undefined with the var 'foo' being a var

michaelherman Mod ➦ Chris B • 21 days ago

Try https://codinglatte.com/pos...

^ | v • Reply • Share ›

Chris B ➦ michaelherman • 21 days ago

Thanks. Strangely, it started using the environment.ts once I went back and rebuilt the images.

^ | v • Reply • Share ›

Marc Kassay • a month ago

> Without the anonymous volume ('/app/node_modules'), the node_modules directory would be overwritten by the mounting of the host directory at run-time.

That is true and I confirmed. Since the bind-mount ".:/app" is binding all contents in the host working directory, this anonymous volume(or volume-mount) is needed. But if we had exclusive bind-mounts that only bound essentials: "src", "angular.json", etc, and omitted "node_modules", this anonymous volume isn't necessary.

So, what advantages are there in having a volume versus bind-mount for our 'node_modules' directory? They both seem to persist in data.

^ | v • Reply • Share ›

michaelherman Mod ➦ Marc Kassay • a month ago

In order to use a bind mount the node_modules directory would have to exist on the host. So, you'd have to install locally, outside of the container first. Then, anytime you want to update the dependencies you'd have to install them locally.

^ | v • Reply • Share ›

Jason Hendee • 2 months ago

> Since we want to use the container version of the “node_modules” folder, we configured another volume: -v /app/node_modules. *You should now be able to remove the local “node_modules”*

reload, but since my local "node_modules" directory is empty, I'm getting red underlines all through my .ts files within VS Code. I could run "npm i" locally, but that defeats the purpose of developing via a container. How do I tell VS Code to use the container's instance of the "node_modules" directory?

BTW: I'm developing on Windows.

^ | v • Reply • Share ›

Marc Kassay ➦ Jason Hendee • a month ago • edited

In an attempt to elaborate where you specified, the following explains theses values in the "Want to use Docker Compose?" section.

```
volumes:
- ".:/app"
- "./app/node_modules"
```

The first item (.:/app) in volumes array, bind-mounts the host machine's current directory into the container's /app folder. As this being a bind type mount, any changes on host machine will reflect into the container. The second item specifies to volume-mount the container's '/app/node_modules' folder. The reason for this mount is to have persistent data, so that when a container has stopped and started again, it will have a built node_modules folder to run off of. Without this volume-mount, the container will fail to launch. Unless there is a populated node_modules folder on the host machine's current directory, which the first item I mentioned about will bind-mount it. But ideally (and as you suggested by saying, "…defeats the purpose…"), you wouldn't want to execute a build on the host machine, as it should be delegated to Docker. Therefore, as Michael suggested, you can delete your local dev instance of node_modules, if it even exists.

see more

1 ^ | v • Reply • Share ›

oli p • 2 months ago • edited

Great article.

So instead of:

```
RUN npm install -g @angular/cli@7.3.9
...
RUN ng build --output-path=dist
```

You can just do:

```
RUN npm run build -- --output-path=./dist
```

Note: the extra dashes (--) are not a typo.

This saves build time and more importantly removes a dependency on a specific angular version.

⌃ | ⌄ • Reply • Share ›

Shekher Khare • 2 months ago

Dude! Your post is reaallly helpful!

⌃ | ⌄ • Reply • Share ›

Fan Jin • 4 months ago • edited

~~Hi, I'm getting `Uncaught TypeError: Cannot read property 'status' of undefined thrown` during docker is running the unit tests. Only one test will fail tho. Do you know anything about this weird behavior?~~ Never mind, it was one of my unit tests. It was passing locally but not during docker test.

⌃ | ⌄ • Reply • Share ›

Ousmane SEYE • 4 months ago

Great tutorial !

Awesome, thanks a lot

Reply • Share ›

Caveman • 6 months ago • edited

Hi, great tutorial. Only thing I have to add is the `default.conf` for nginx to work correctly with Angular Routing. I added the nginx config in the repo and copy it to the container in the `Dockerfile-prod`, I would like to hear your opinion on this approach.

Reply • Share ›

michaelherman Mod ➦ Caveman • 5 months ago

Yes, that's exactly how I do it.

Reply • Share ›

Mário De Sá Vera • a year ago

A quick fix as the phrase "as a builder" at the production Dockerfile should be a comment.

Reply • Share ›

michaelherman Mod ➦ Mário De Sá Vera • a year ago • edited

What are you referring to?

"as builder" from `FROM node:9.6.1 as builder` should NOT be a comment

1 • Reply • Share ›

Mário De Sá Vera ➦ michaelherman • a year ago

You are very correct Michael , sorry for that as my docker release is out of date and did not parse correctly this "as [image name]" new feature. Anyway it is here for other folks to read and update their docker version as well.

1 • Reply • Share ›

santosh chougule • a year ago

Hi ,

```
-v ${PWD}:/src/app \
-v /src/app/node_modules \
-p 4200:4200 \
--rm \
something-clever
```

∧ | ∨ • Reply • Share ›

michaelherman Mod ➦ santosh chougule • a year ago

Try exposing port 49153 -> https://stackoverflow.com/q...

1 ∧ | ∨ • Reply • Share ›

Sam Wilson ➦ michaelherman • 7 months ago • edited

Still doesn't work for me after adding "EXPOSE 4200 49153" to the docker file.
The error message is "Cannot read property 'config' of null" (before and after adding expose).
The command to run the container in the background doesn't return any errors, but "docker exec -it ..." command says the container isn't running.
The "ng e2e" command works fine and opens the site in chrome for a brief moment.

∧ | ∨ • Reply • Share ›

michaelherman Mod ➦ Sam Wilson • 7 months ago

That error means it can't find the Angular CLI config. See if this helps -> https://github.com/angular/...

5 ∧ | ∨ • Reply • Share ›

hmartini ➦ santosh chougule • 6 months ago • edited

Late, but may be others have the same problem.

Edit your Dockerfile:
CMD ng serve --host 0.0.0.0 --port 4200 --disableHostCheck=true --poll 100

I haven't tested, if you can left the --port and --disableHostCheck option, but the --poll is

I use this command in a vagrant/docker environment.

^ | ⌄ • Reply • Share ›

David Tang • a year ago

Is the "ng serve" executing from within the container? If I understand this correctly, "ng new something-clever" was run on outside of the container, which means we would need to install node and angular-cli on both the host and in the container right?

^ | ⌄ • Reply • Share ›

michaelherman Mod ➦ David Tang • a year ago

`ng-serve` is running within the container.

You are right in that I created the initial Angular app - via `ng new something-clever` - outside the container. The purpose of that is to just get an Angular app created. You don't necessarily need to do this, though. Maybe you already have an app locally (like you cloned it down from GitHub). Or you can even use Docker to scaffold out an app within the container, by adding the `ng new something-clever` to the Dockerfile before running `ng-serve`.

Hope that helps.

1 ^ | ⌄ • Reply • Share ›

David Tang ➦ michaelherman • a year ago

Thanks! Have you found running "ng serve" and live reload slower at all, as opposed to just running it outside of the container on your host machine?

1 ^ | ⌄ • Reply • Share ›

michaelherman Mod ➦ David Tang • a year ago

It can be. Try https://github.com/angular/...

1 ^ | ⌄ • Reply • Share ›

Nathan Johnstone ➦ David Tang • 4 months ago

That would probably be the case for Windows, since you have to remember

^ | ˅ • Reply • Share ›

ukwala supermarket • a year ago • edited

Hello ,awesome stuff was finally able to serve the app but live reload doesnt work ,any tips ?
FYI i used docker-compose.yml

^ | ˅ • Reply • Share ›

Nathan Johnstone ➦ ukwala supermarket • 4 months ago • edited

I'm using windows and to get the live reload working you need make sure you have the environment variable in that is mentioned in the docker compose

CHOKIDAR_USEPOLLING=true

This also is true if you run it directly with "docker run", you need the env var to enable live reload

docker run -it -v ${PWD}:/app -v /app/node_modules -p 4201:4200 --rm -e "CHOKIDAR_USEPOLLING=true" example:dev

I've also added "-it" to make sure that ctrl-c is handled correctly and the container shutsdown and does not keep executing in the background.

Great work Michael, this is a great guide to get up and running quickly though !

^ | ˅ • Reply • Share ›

michaelherman Mod ➦ ukwala supermarket • a year ago

Which versions of Angular and Docker are you using?

^ | ˅ • Reply • Share ›

ukwala supermarket ➦ michaelherman • a year ago

Switched from windows 10 to Linux, installed some few dependencies ,now am set .
Thanks for the follow up

1 ^ | ˅ • Reply • Share ›

I keep on getting : no space left on device
in step 8/9

Step 8/9 : COPY . /usr/src/app
ERROR: Service '...' failed to build: Error processing tar file ... no space left on device

tried searching for it but nothing i found worked.. is there a missing cleanup step ?

Reply • Share ›

LB ➦ LB • a year ago

this worked for me so far, in case someone faces same problem:

docker volume ls -qf dangling=true | xargs -r docker volume rm;

1 • Reply • Share ›

michaelherman Mod ➦ LB • a year ago • edited

You can also do "docker system prune". https://docs.docker.com/eng...

Reply • Share ›

Guy • 2 years ago

Instead of building from node:9.6.1 and installing Chrome you can use another image with node, chrome and firefox pre-installed. https://hub.docker.com/r/di...

Reply • Share ›

michaelherman Mod ➦ Guy • 2 years ago

Sure. One should create and manage this image on their own so that they have control over the versions used rather than extending from your image, though.

I don't like this approach since I like to change the versions on the fly.

1 • Reply • Share ›

Guy ➦ michaelherman • 2 years ago

Oh, no doubt. In a perfect world we would have specific version numbers for each and

1 ˄ | ˅ • Reply • Share ›

✉ Subscribe Ⓓ Add Disqus to your siteAdd DisqusAdd 🔒 Disqus' Privacy PolicyPrivacy PolicyPrivacy

Questions? michael at mherman dot org

Back to top