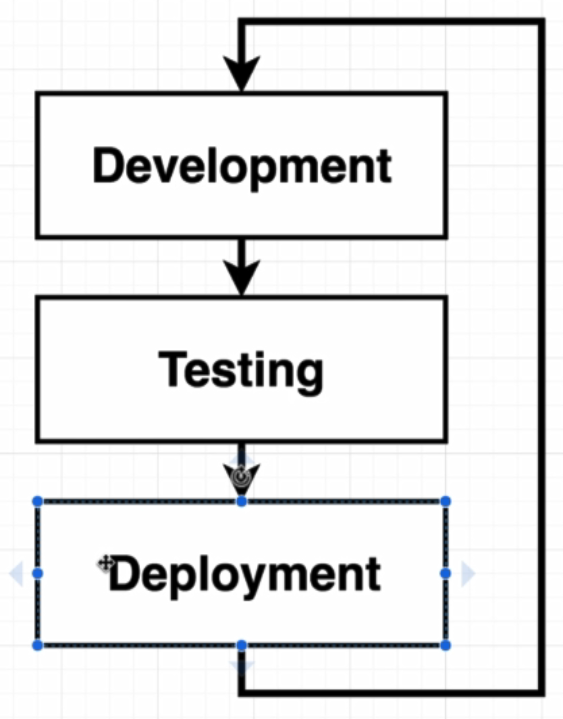
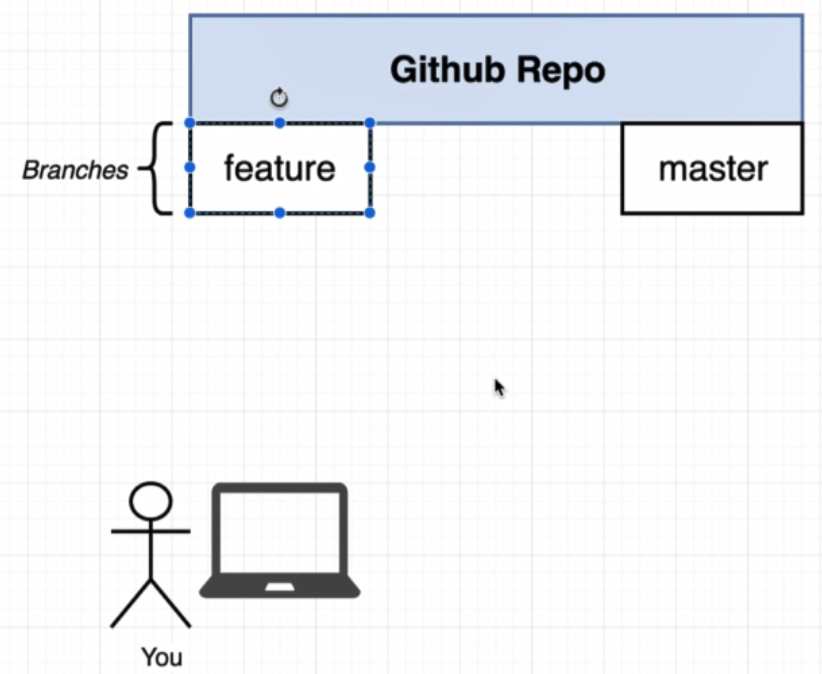
**Creating a Production-Grade Workflow**

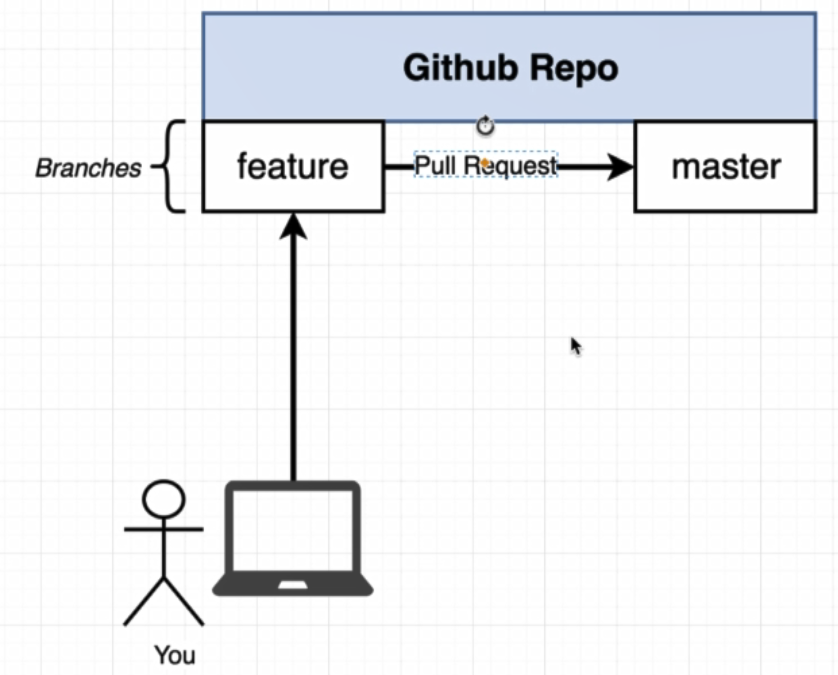
In this section we will work on following diagram:



Our development workflow is going to revolve around creating a github repository. In the repository we have two branch as bellow:

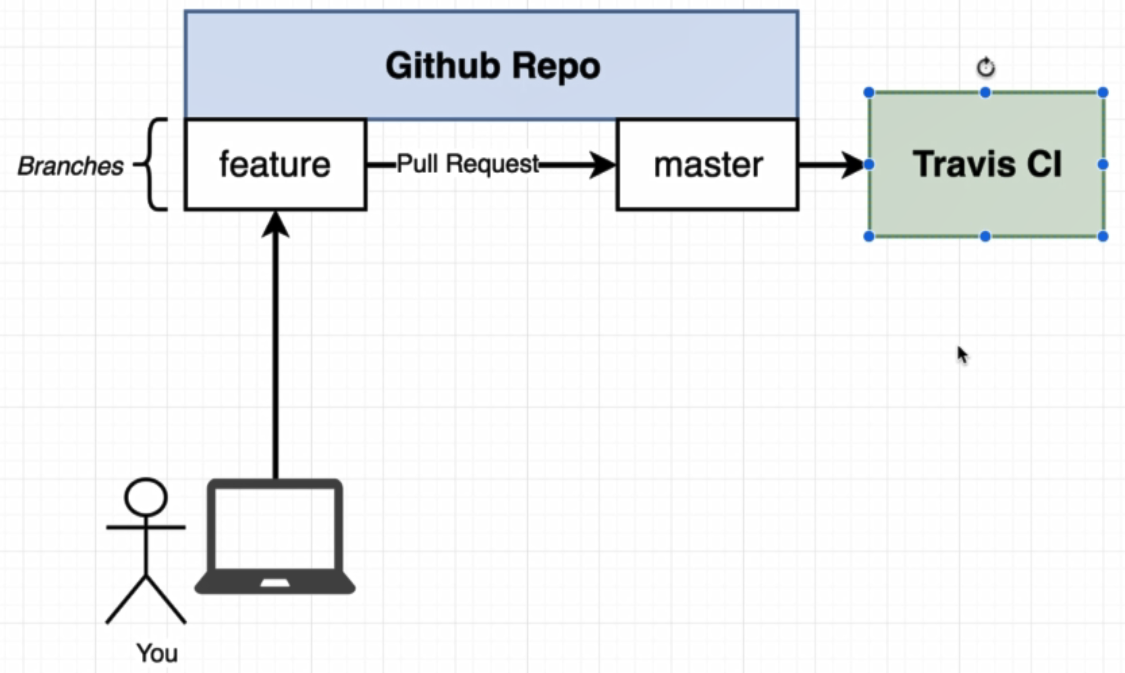


* Feature Branch
  + A branch that we are going to add some code
  + We are only pull and push code from feature branch.
* Master Branch
  + Represent very clean working copy of our codebase.
  + **Any changes on master branch will automatically deployeing to our hosting provider.**



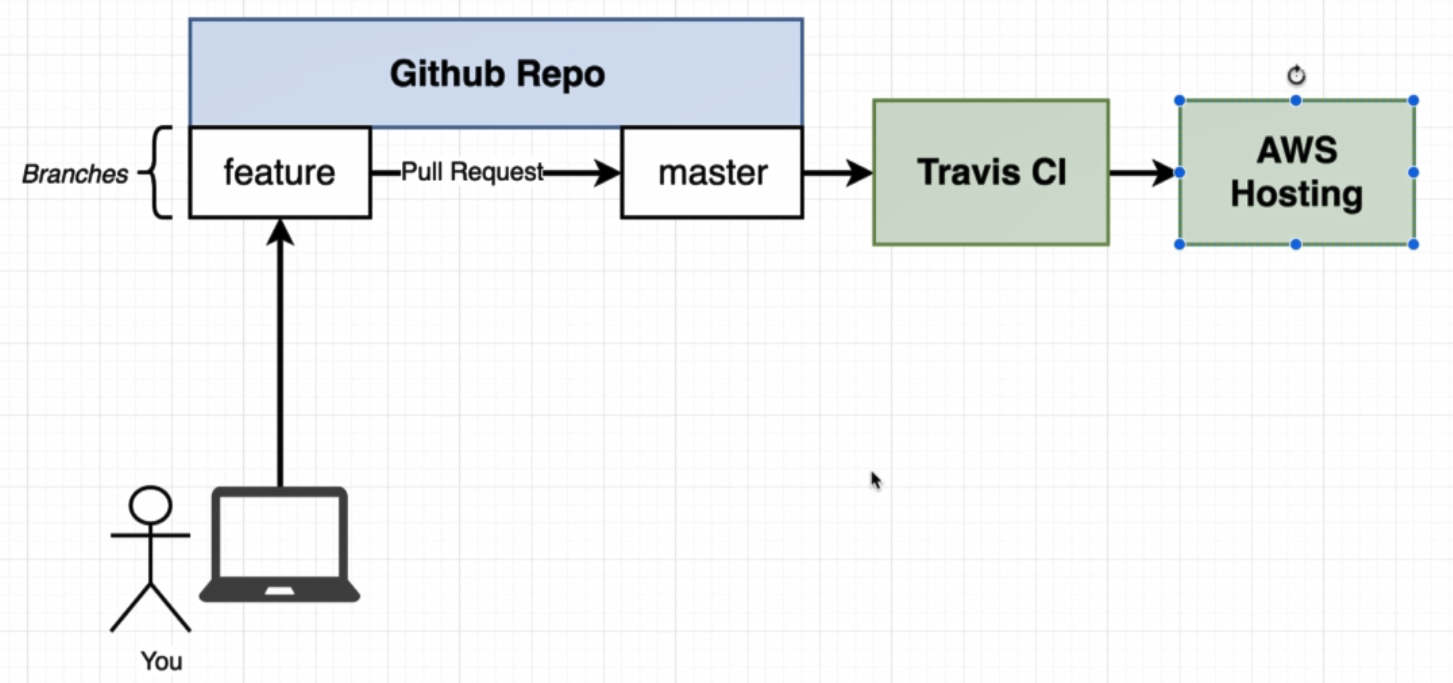
To merge our feature branch to master branch we will create a Pull Request.

After this step we will set up a workflow to automatically take our application and push it over Travis CI.

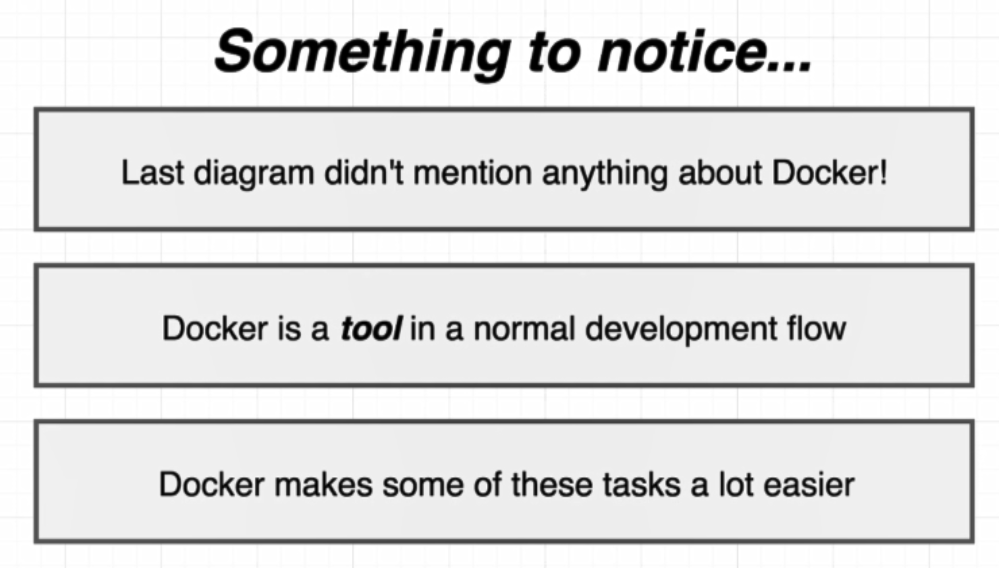


Travis CI is a countinuous integration provider. It pull down our code and run a set of tests that we are write on our codebase.

The last step is that Travis CI get our project and push it over some Amazon Web Service hosting.



What is the docker role in this senarios.



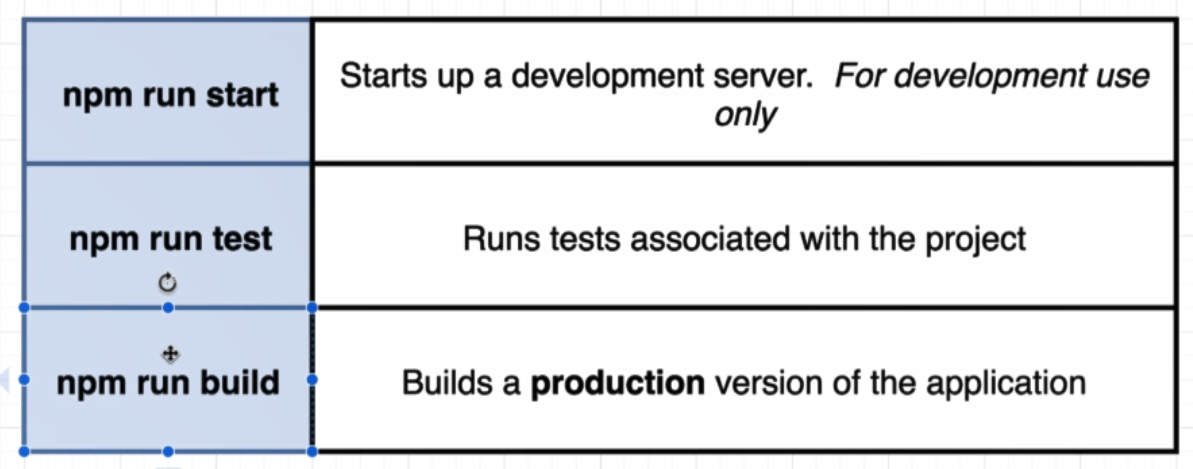
To work on previously described workflow we have to has node.js on our machine, after that install a package to create react project:

* Npm install –g create-react-app

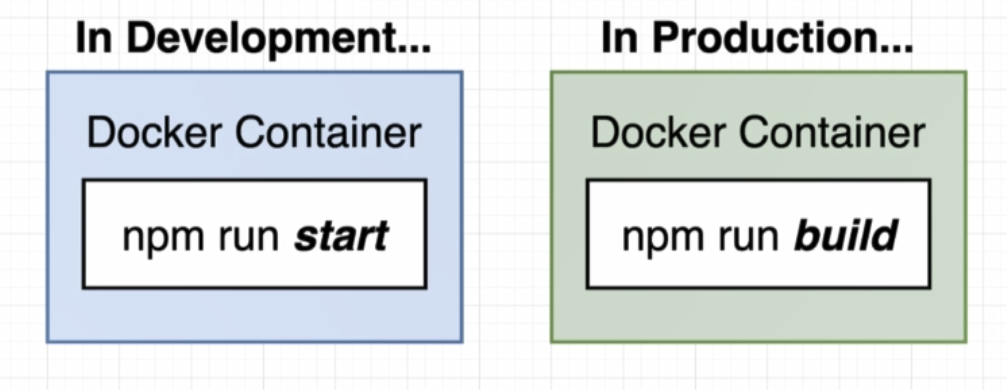
To create react project use command bellow:

* Create-react-app frontend

Three command that we will use on project:



We will have two seperate dockerfile on created project as bellow:



**Docker Container in development:**

* Create a Dockerfile.dev file inside project for development container.
* #Dockerfile.dev
* FROM node:alpine
* WORKDIR '/app'
* COPY package.json .
* RUN npm install
* COPY . .
* CMD ["npm", "run", "start"]

**To run docker with coustome name:**

* docker build –f Dockerfile.dev .
* **docker build . will look for a file with name Dockerfile, but for build coustome file we should use command above.**

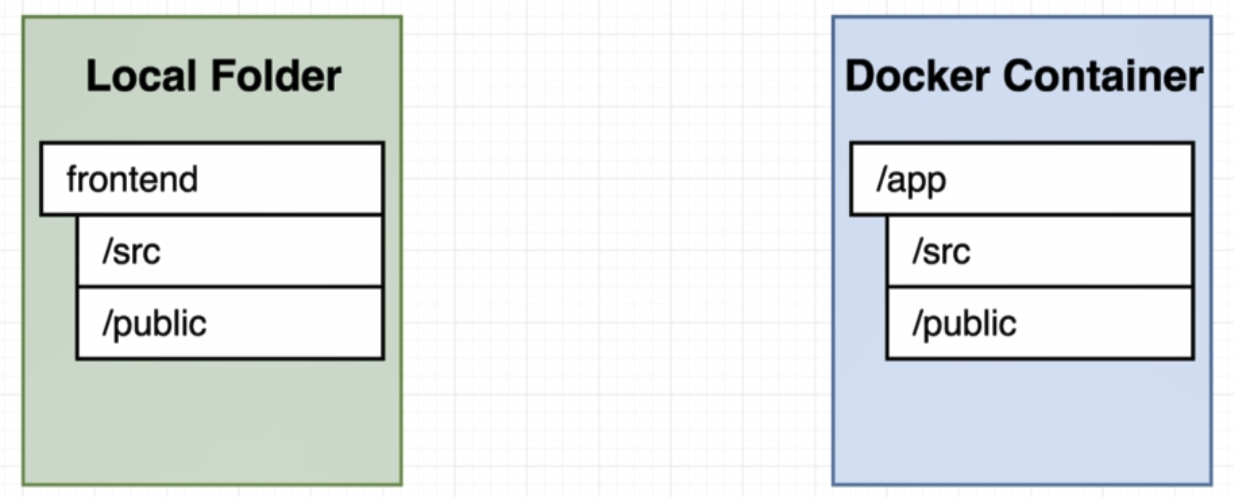
By building Dockerfile.dev, node\_modules folder will copy inside container and after “RUN npm install” this folder will bi duplicated. To avoid this we can delete node\_modules folder from project because inside container it will automatically created.

In building dockerfile dependencies will be install for first time. After building doockerfile we can run it by using command :

* docker run –p 3000:3000 ID

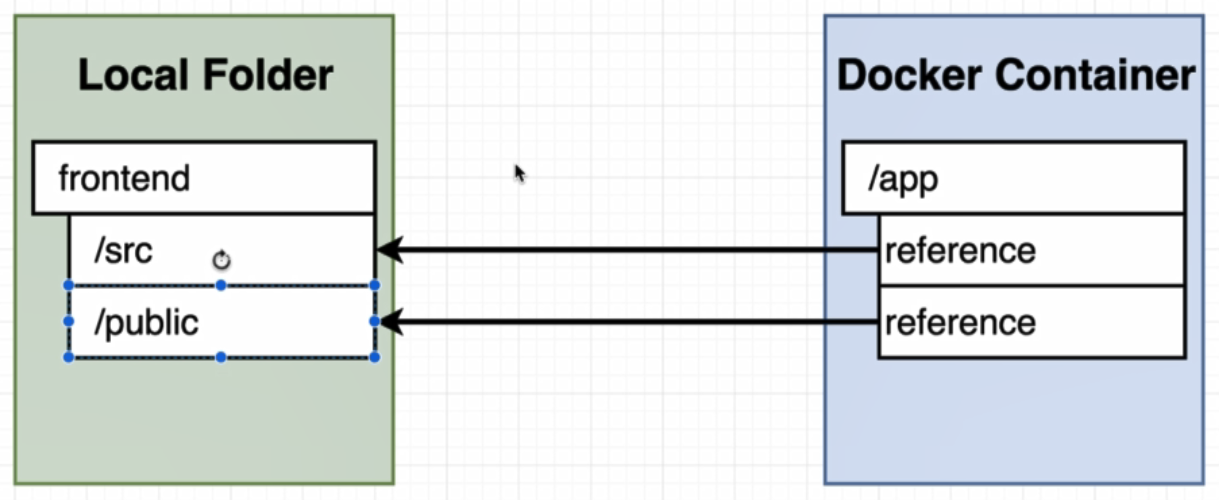
**On every changes on code docker have to COPY fn snapshot of all files from local to inside container. To avoid build on every code changes we can use method described bellow.**

By using previouse steps we have container as shown bellow. This container is a temprorary container created during image build.

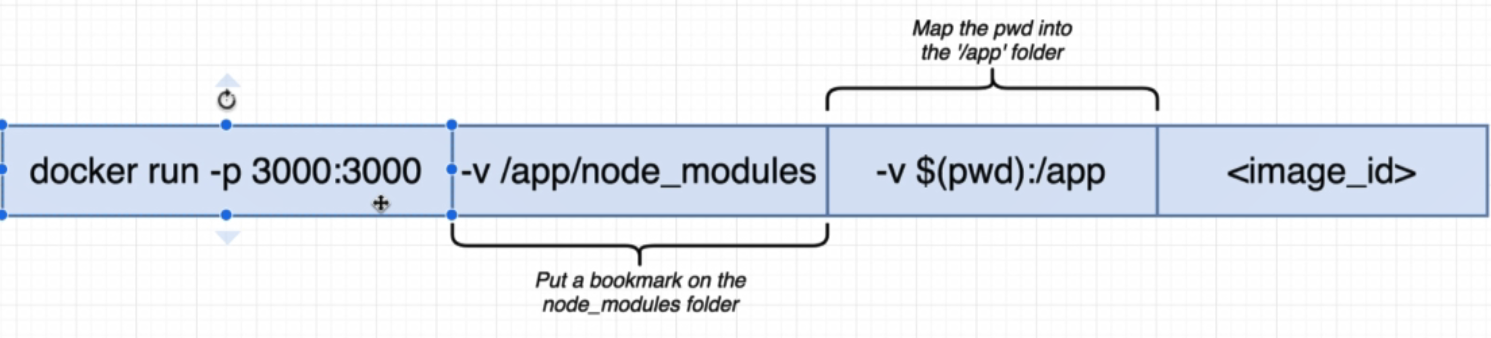


**Volume:**

We essentially set up a placeholder inside of our docker container. We are no longer going to copy source directory. Instead of that we are going to put some sort of references inside container to source code. (This action is like port mapping)



To run docker with volumes:



**Map the pwd into the ‘/app’ folder**:

This command say that take everything inside of our present working directory and map it up to the app folder inside of our container.

* -v is used to set up a volume
* $(pwd):/app
  + Set Present Working Directory
  + pwd : get the current directory path

**Put a bookmark on the node\_modules folder:**

Because of the node\_modules will be inside container and there is no need to reference to local machine we specify that do not try to map [folder] against anything else.

* -v is used to set up a volume
* Path after –v specify that there is no need to map this address to anything else.

Instead of $(pwd) we can give an current directory like d:/frontend

* **docker run –p 3000:3000 –v /app/node\_modules –v $(pwd):/app dd41355e846a**

**On windows machine for react project we have to create .env file and write “**CHOKIDAR\_USEPOLLING=true**” inside it other wise changes on project will not detected on browser.**

Now in this section we are going to create docker-compose file for complex docker run command described before.

version: '3'

services:

web:

build: # Refer to Dockerfile created for image

context: . # Specify where we want all the files and folders for this image to be pulled from

dockerfile: Dockerfile.dev # Use file to build the image

ports:

- "3000:3000"

volumes:

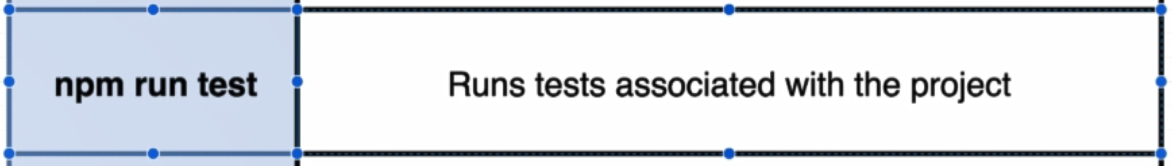
- /app/node\_modules # Do not try to map a folder up against f node\_modules

- .:/app # Map the current folder that contain source code to app folder inside container

In the last section we solid infrastructure in place to run our container in a development enviroment. So the action that described before don:



Now we are going to running he tests inside of our container:



To run test for react application on local machine we can use command bellow:

* docker run ID npm run test
* docker run –it ID npm run test

To run test command inside docker container we can add another service inside docker-compose.yml file.

By change docker-compose.yml as bellow we will have two container one for web and another one for test

version: '3'

services:

web:

build:

context: .

dockerfile: Dockerfile.dev

ports:

- "3000:3000"

volumes:

- /app/node\_modules

- .:/app

tests:

build:

context: .

dockerfile: Dockerfile.dev

volumes:

- /app/node\_modules

- .:/app

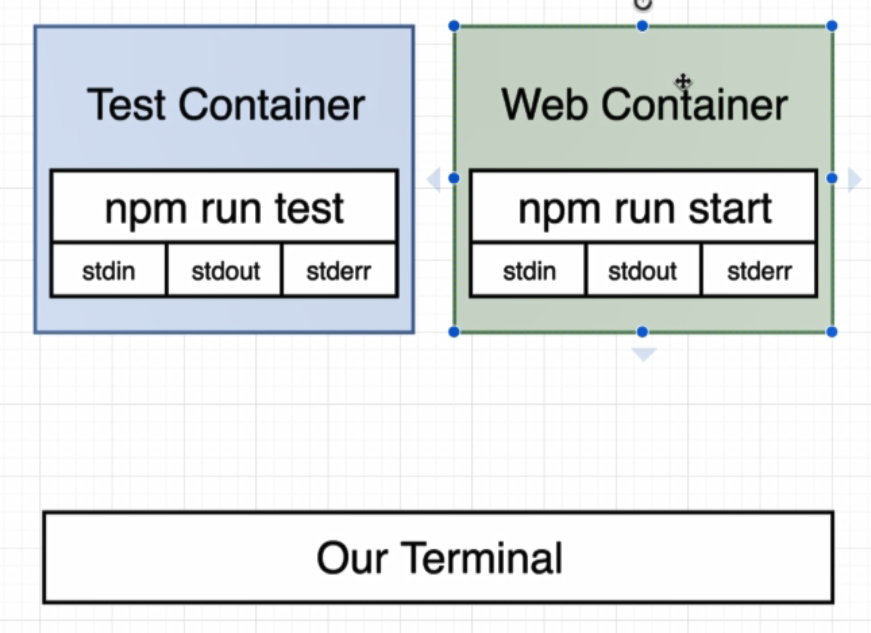
command: ["npm", "run", "test"]

Now we can run and build docker-compose by using command bellow:

* docker-compose up –build
* docker-compose up

The downside of this approach is that we are getting all the output from our test suite inside of the kind of logging interface of docker compose. We cannot get an input from command to test container.

The diagram bellow display what actually we did in this section

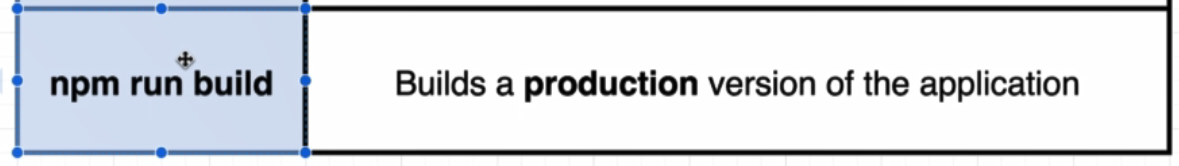


Both this container start up with a primary command.

* npm run test (For Test Container)
* npm run start (For Web Container)

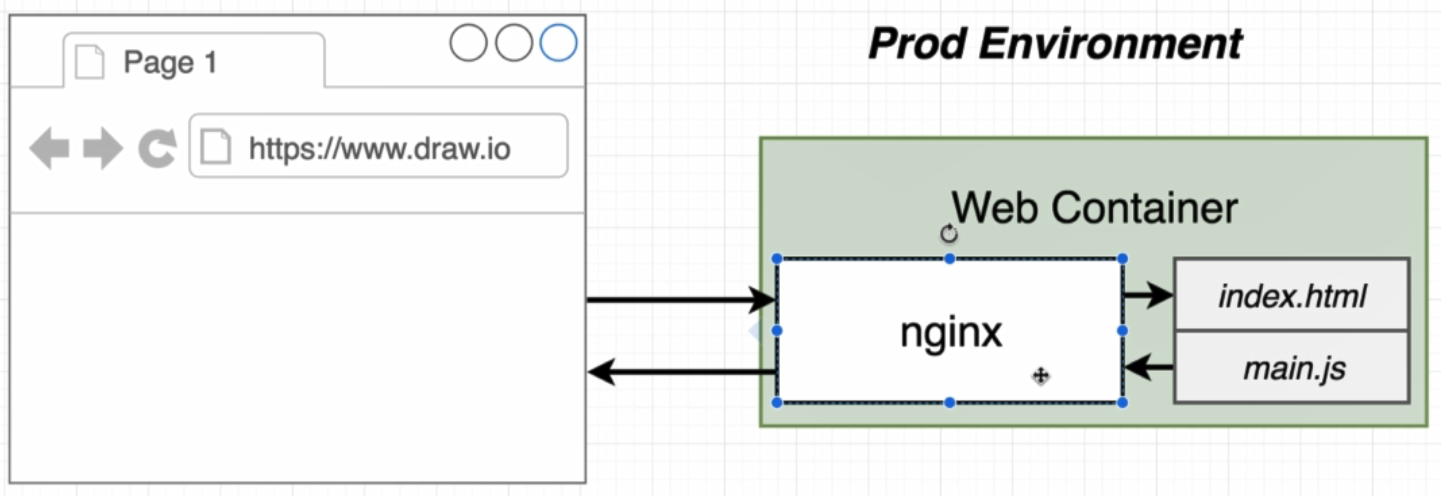
If we write a command in our terminal it is not go through to any container by default. By using docker-compose it is not easy to do that. There is a alternative way to do this by usind **docker** **exec.**

The last to section done and now it is time to **npm run build**



This command used to build a production version of the application. Essentially takes all the javascript files, process them and put these files in a folder on your hard drive.

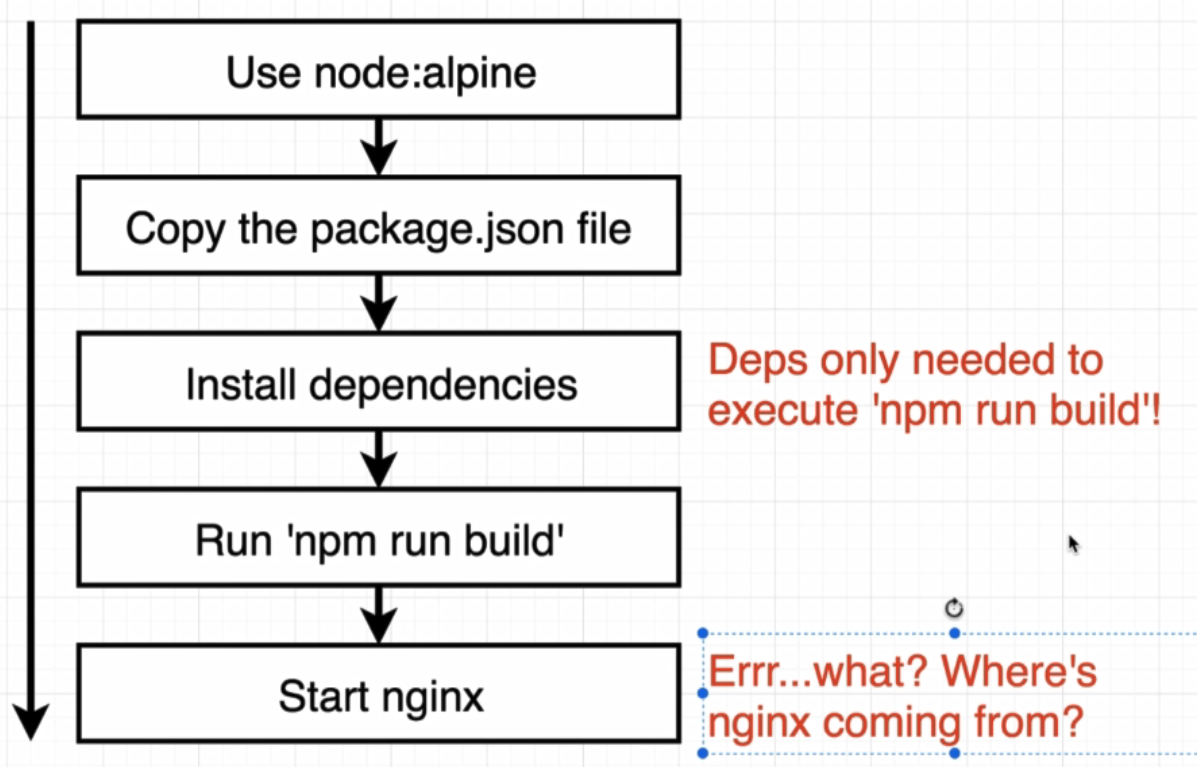
To create production environment to response to incomming request from client and response them with files inside it we can use nginx web server:



It just taking incomming trafic and somehow routing it or responding it with some static files.

To do this first we create a another dockerfile inside project with name **Dockerfile**

Inside the dockerfile to create image we will do the steps that shown in bellow diragram:

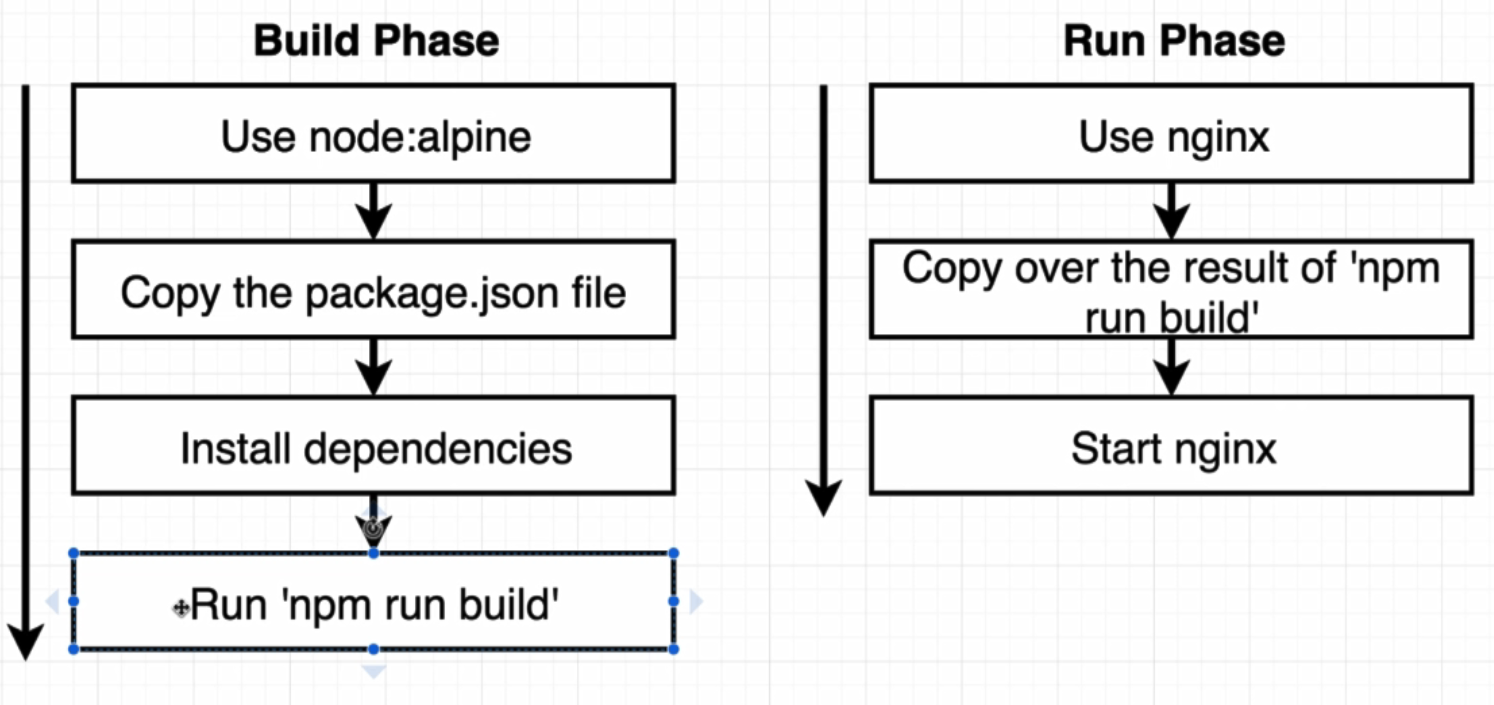


Install dependency only used on **npm build,** after this action there is no need to package inside node\_modules and a build folder will not contain node\_modules folder too.

Another big issue with this flow is the start nginx step. Where the nginx come from , when we installed or configured or set up in any shape or form.

There is a nginx image on hub.docker and it can be probably used to host some simple static content.

We are going to build a dockerfile that has something called a multi step build process. In the docker container we are going to have two different locks of configuration.



The result of **Build Phase** is build folder of our web project.

In **Run Phase** it will copy result of **Build Phase**(build folder) inside it. İn this case all dependencies for node:alpine and npm package will be droped from result folder and this is what we want.

The contain of the created Dockerfile shown as bellow:

FROM node:alpine as builder

WORKDIR '/app'

COPY package.json .

RUN npm install

COPY . .

RUN npm run build

# result inside build folder will be placed inside /app/build of container

FROM nginx

#Copy files from /app/build of builder container to this container to /usr/share/nginx/html folder

COPY --from=builder/app/build /usr/share/nginx/html

#The default command of the enginx container is "start nginx" so there is no need to write that

After create a Dockerfile as displayed above, by running command bellow we can run docker and mapping port to it.

* docker build .
* docker run -p 8080:80 35f09d7a7f9c

**In the next section we will work with Github, Travis CI and AWS on docker.**