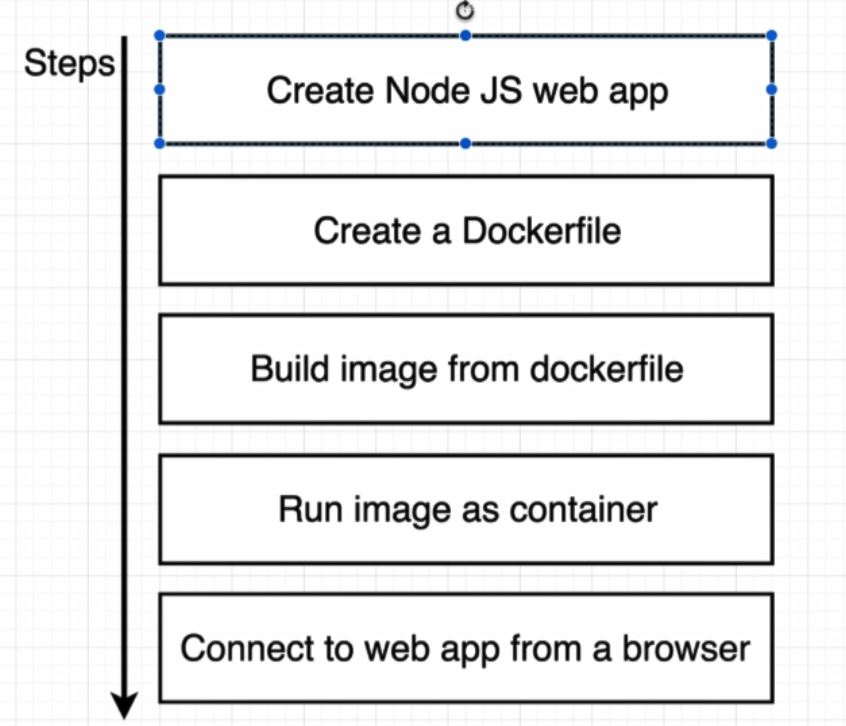
**Making Real Projects with Docker**

In this section we follows the steps bellow to create real docker project to run Node.js project dependency on it.



We are going to make dockerfile to create an image that is going to be used to run our web server inside of container.

Create Node JS web application:

* Create core of the node.js web application

To start up created application:

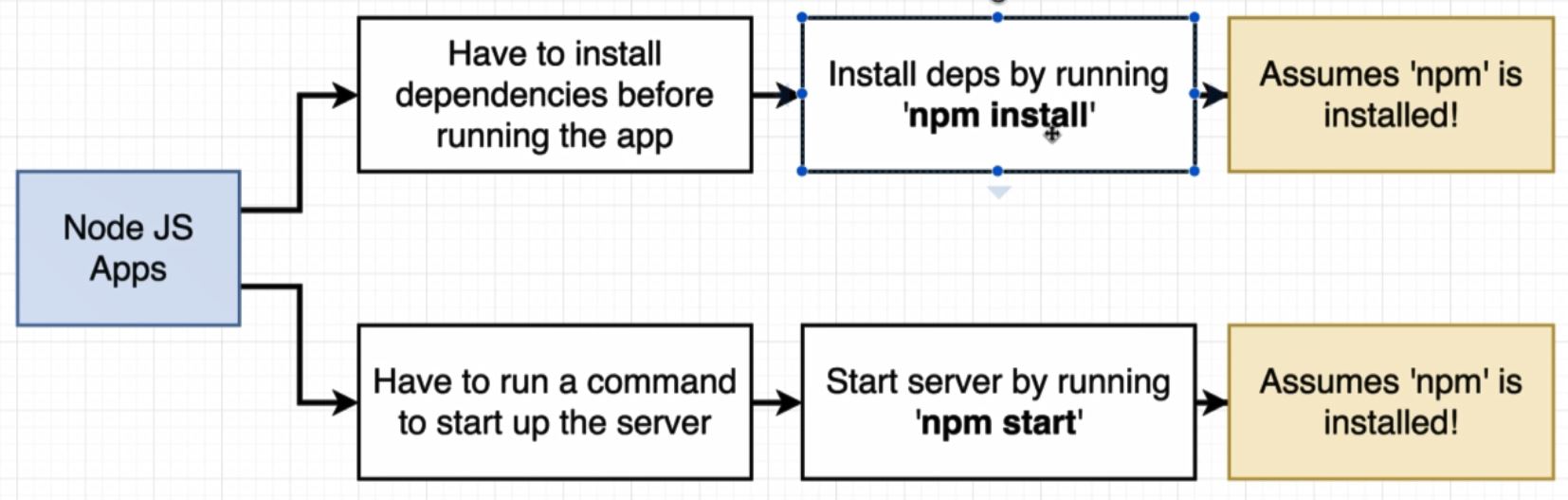
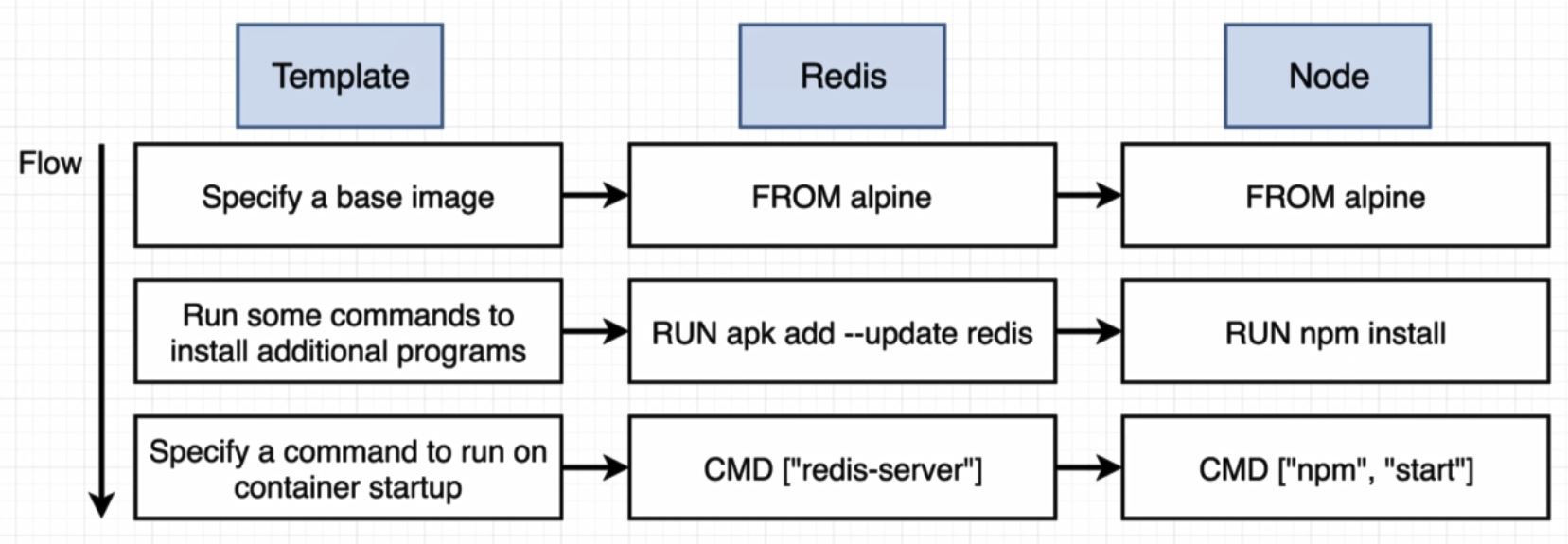


Image shown above display steps that should be pass to execute web app, to create image to do this we will do as like latest section:



İnside Dockerfile:

# Specify a base image

FROM alpine

# Install some dependencies

RUN npm install

#Default command

CMD [ "npm", "start" ]

By running

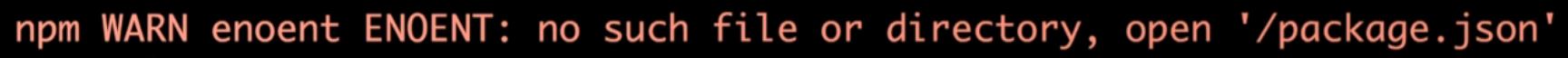
* docker build .

we will get an error: npm not found. This error says that alpine has not have npm command on it. To sole this error we have two way:

1. we can try to find a base image that already has node and npm pre-installed inside of it.
2. Or we can countinue using alpine and run additional command to attempt to install node and npm inside of our image.
3. Find repository from hub.docker.com
   * Node repository is a docker image that has node pre-nstalled on it.

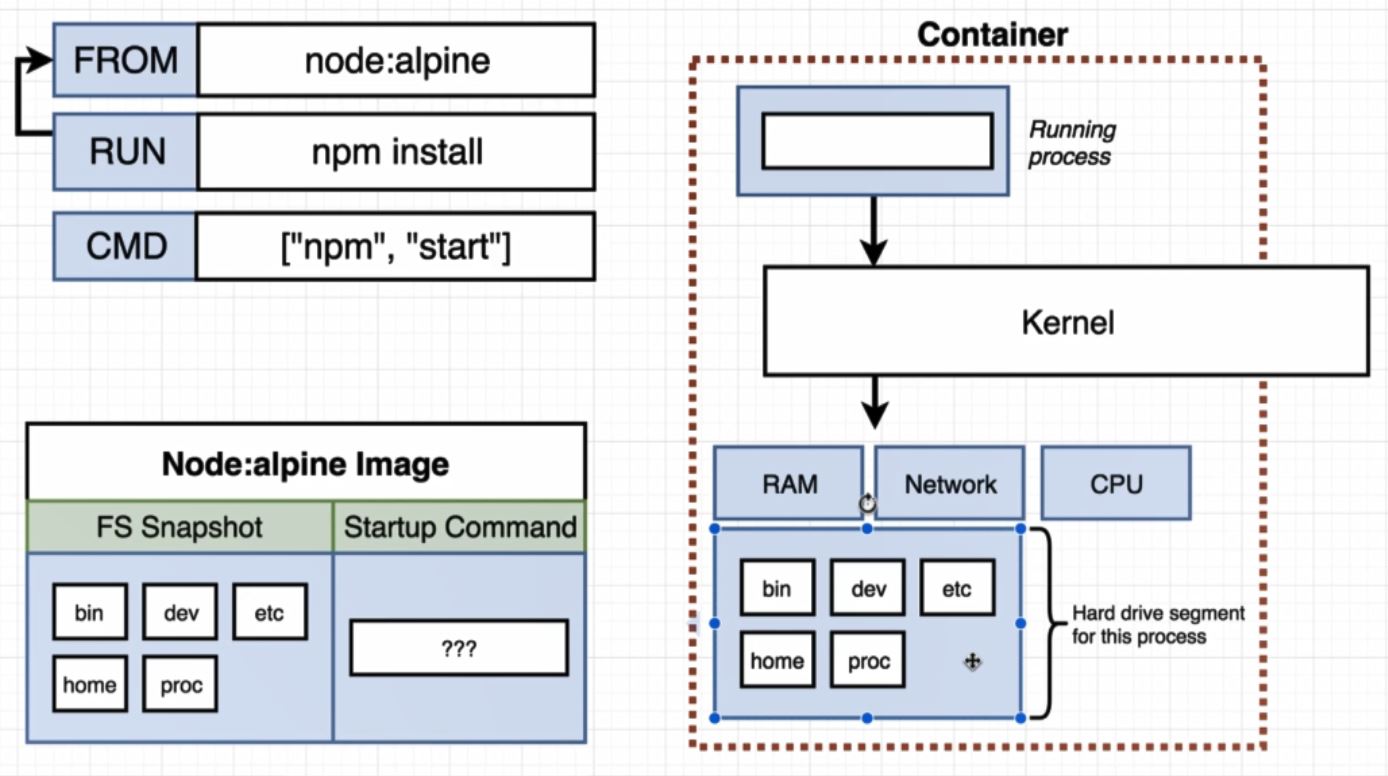
FROM node:alpine

After rebuild dockerfile, “npm not found” error will be fixed but now we have another error as shown bellow:



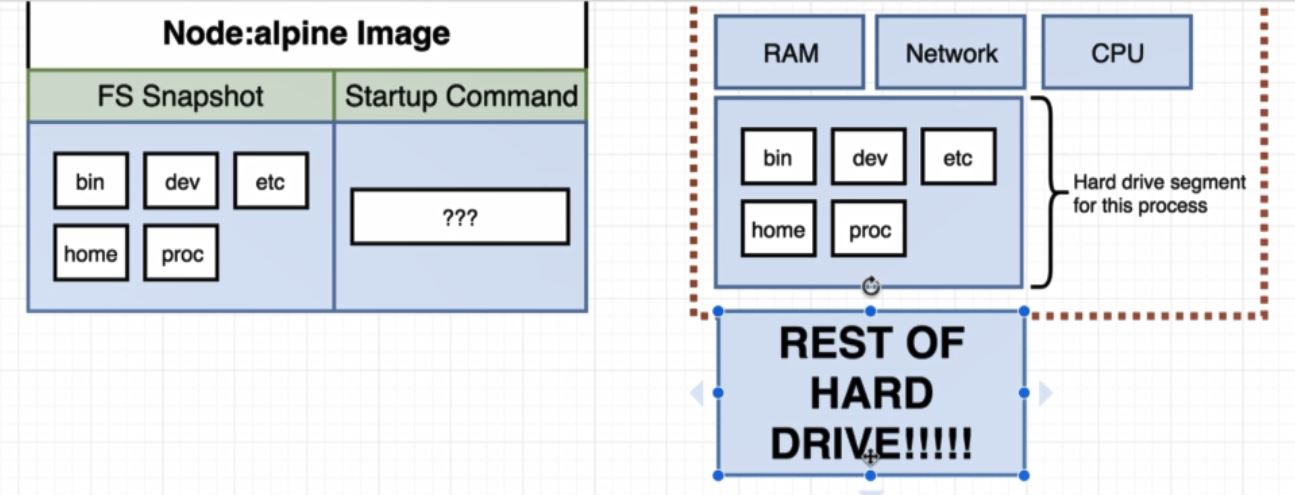
Actually we have package.json inside same folder but as we decribe before, when we build dockerfile, in every step docker create temporary container and then remove it.

İn step “npm install” docker structure is like bellow:



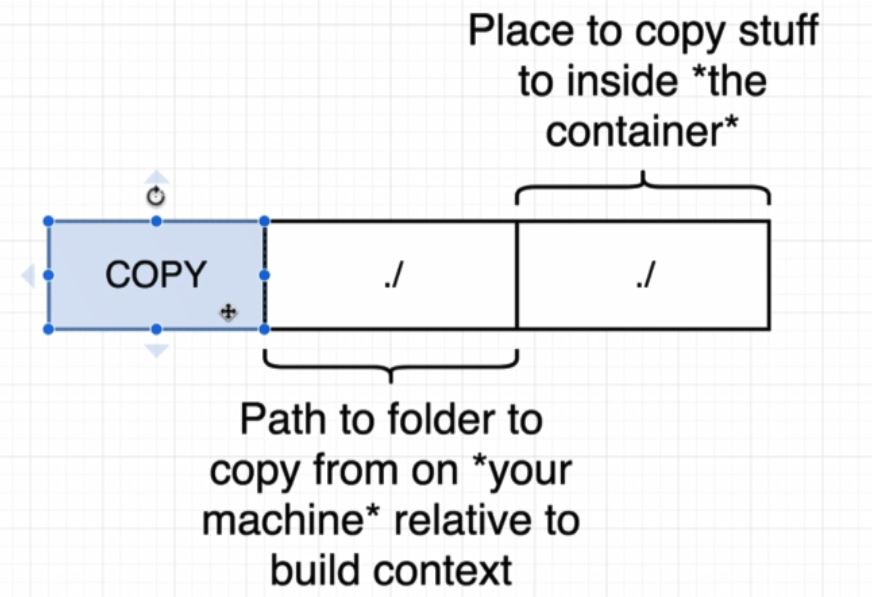
By running “npm install”, npm look for package.json file inside Node:alpine image (not over folder) and as shown above there is no file with name package.json inside FS Snapshot.

Remember we are communicate with segment of hard drive inside container,



Over package.json file exist in REST OF HARD DRIVE.

To copy file or folder from our local file system inside temporary file system that is created during the build process we can use command bellow:



# Specify a base image

FROM node:alpine

# Install some dependencies

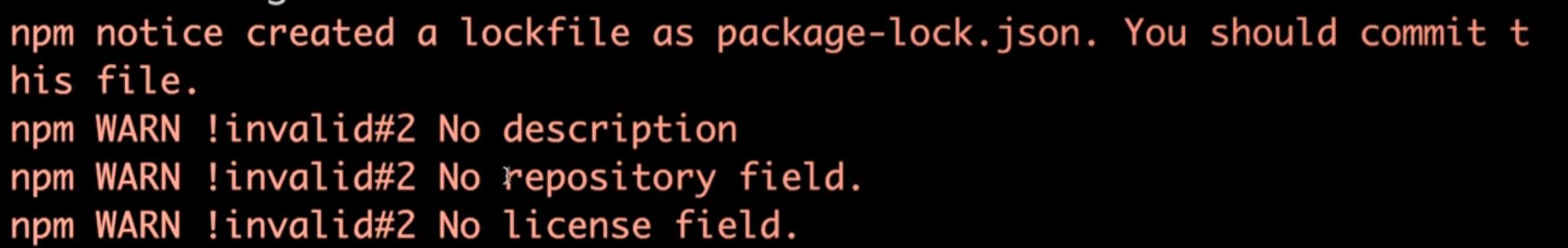
COPY ./ ./

RUN npm install

#Default command

CMD [ "npm", "start" ]

After rebuild dockerfile we get warning as bellow but its not important for now:



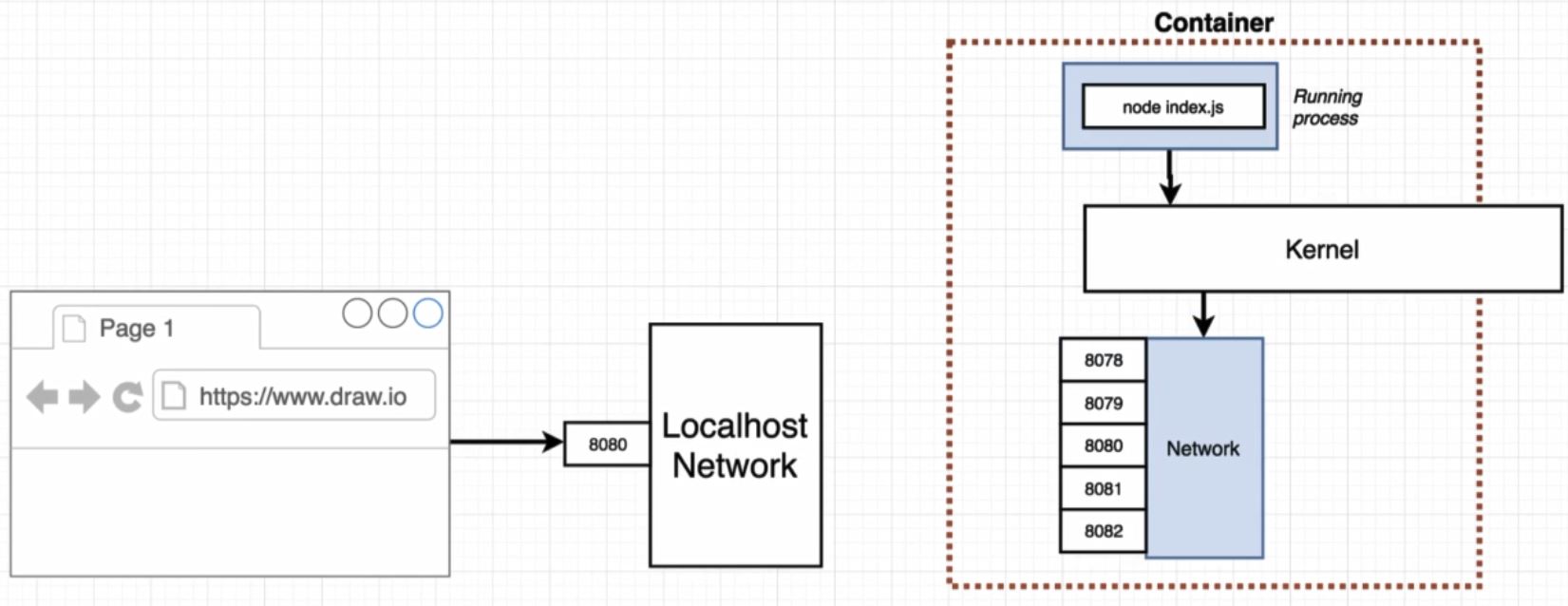
Now we are going to tag image

* docker build –t tohid1987/simpleweb .

Now we can run docker container

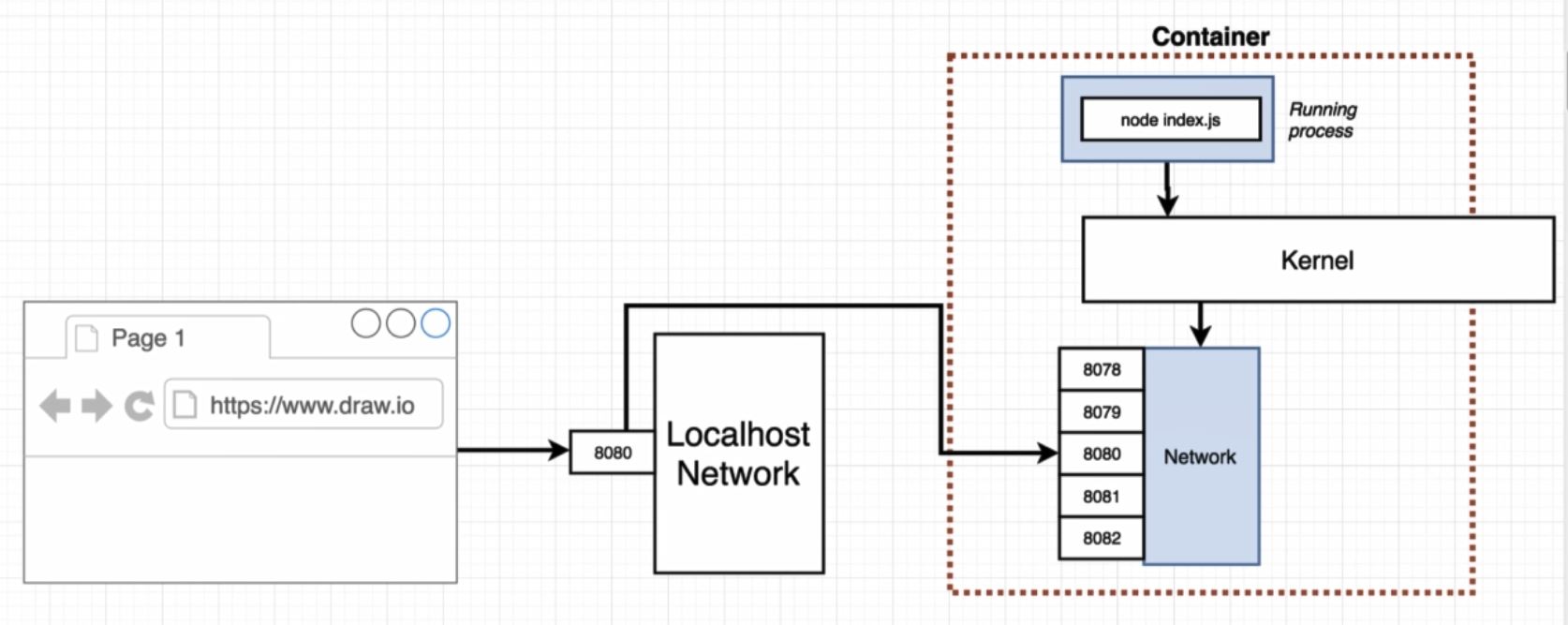
* docker run tohid/simpleweb

But we cannot reach to localhost:8080 it because that there is no trafic between local host network is routed into the container.



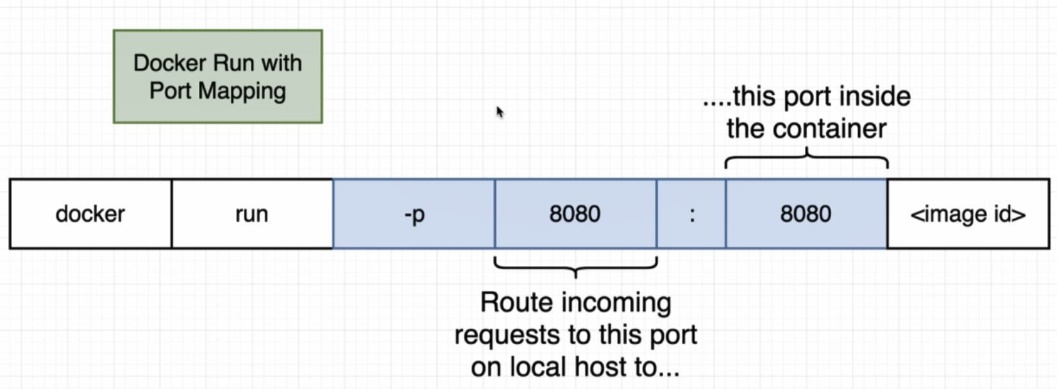
To sole this problem we have to set up a explicit port mapping:

Port mapping means that any time any request come to localhost:8080 take request automatically and forward it to some port inside container.



There is only limitation on the ability of incoming traffic to get in to the container, there is no limitation to outcoming trafic of container. Container can communicate with internet with out limitation.

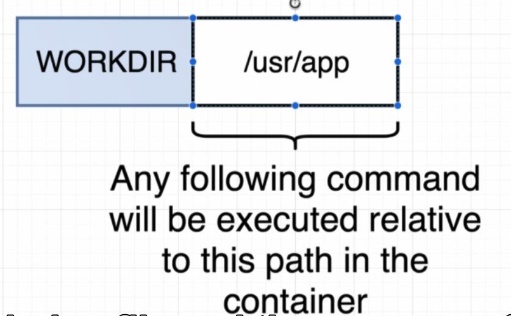
We cannot solve this problem inside dockerfile. Because it is a runtime constraint. We have to map inside run command as bellow:



* docker run –p 8080:8080 tohid1987/simpleweb

Now we can browe to localhost:8080 and see the result.

To chage work directory inside container we can use command as shown bellow:



By default work directory is set to root but it is not good idea because if there is a file or directory with same name the replace action will be acure.

# Specify a base image

FROM node:alpine

WORKDIR /usr/app

# Install some dependencies

COPY ./ ./

RUN npm install

#Default command

CMD [ "npm", "start" ]

Now when we browse to localhost:8080 we can see the index.js content(hi there)

İf we change the “hi there ” to “bye there” and save it,the change will not affect the localhost:8080, because it work with snapshot of last index.js(copy ./ ./)

To solve this problem **we will have to rebuild the container** and do the action done in last section. In this situation npm install will reinstall again and take much time to done. To avoid npm inistall on every rebuild container we can change Dockerfile as bellow:

COPY ./package.json ./

RUN npm install

COPY ./ ./

In second build of container, lines befoe COPY ./ ./ will not be change and there will be no need to reinstall npm install, just it will copy all file and folder from local inside container.