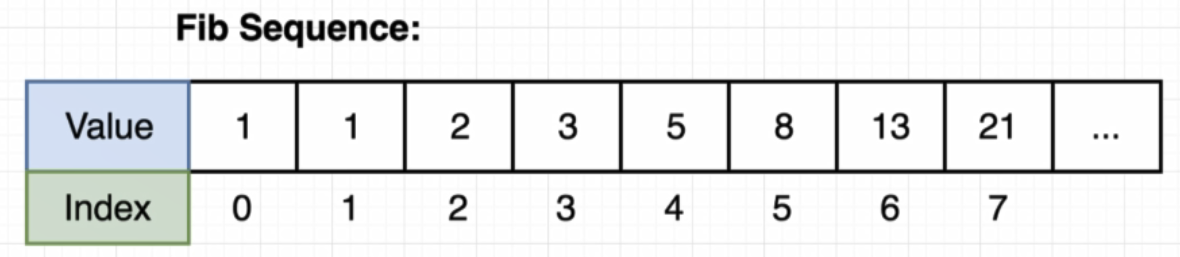
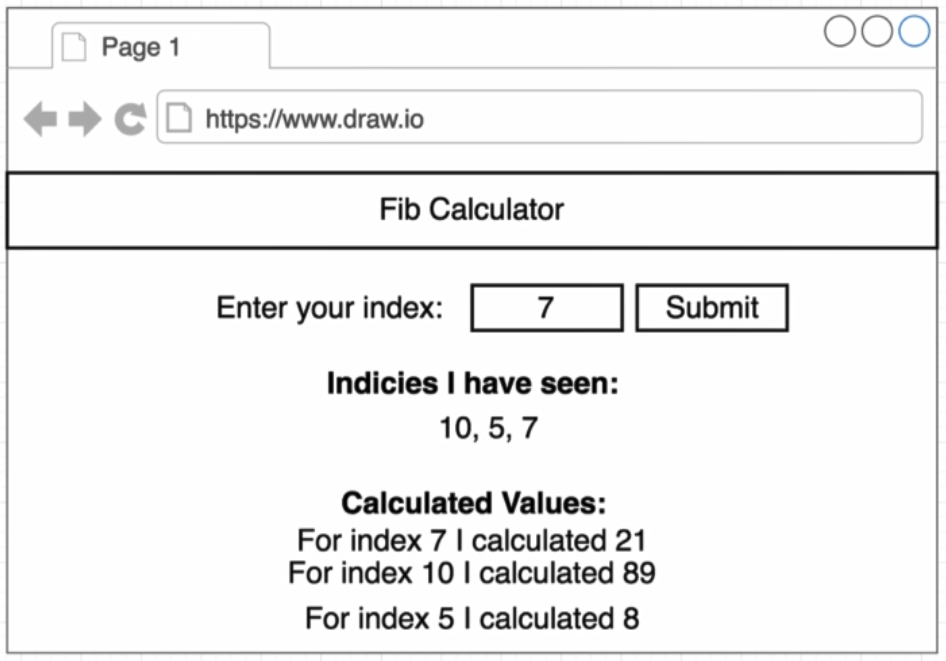
**08 Building a Multi-Container Application**

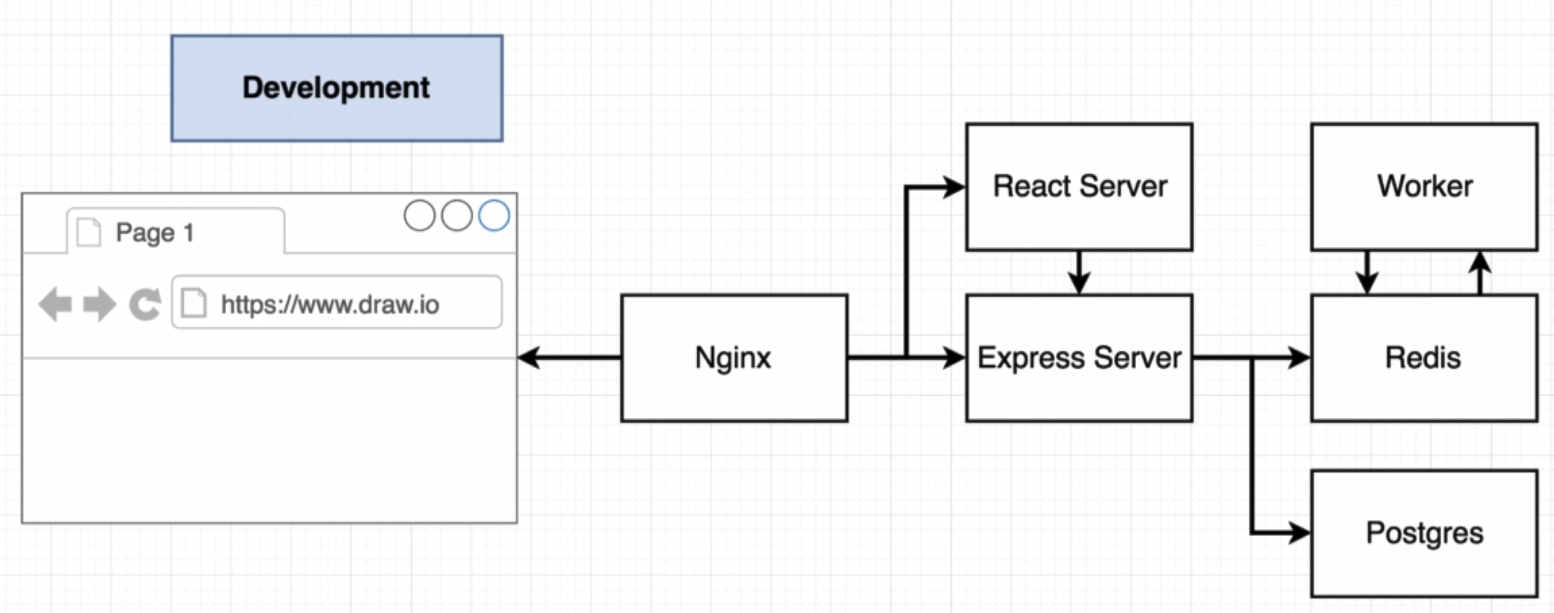
In this section we will write an application to calculate Fibonacci based on Index:



We will make a page that take input as Index from user and calculate fib of given index. After that we will keep the result in memory.



Backend architecture:



* Values **Indices I have seen** section will stored in Postgre.
* **Calculated values** will be saved on redis database.

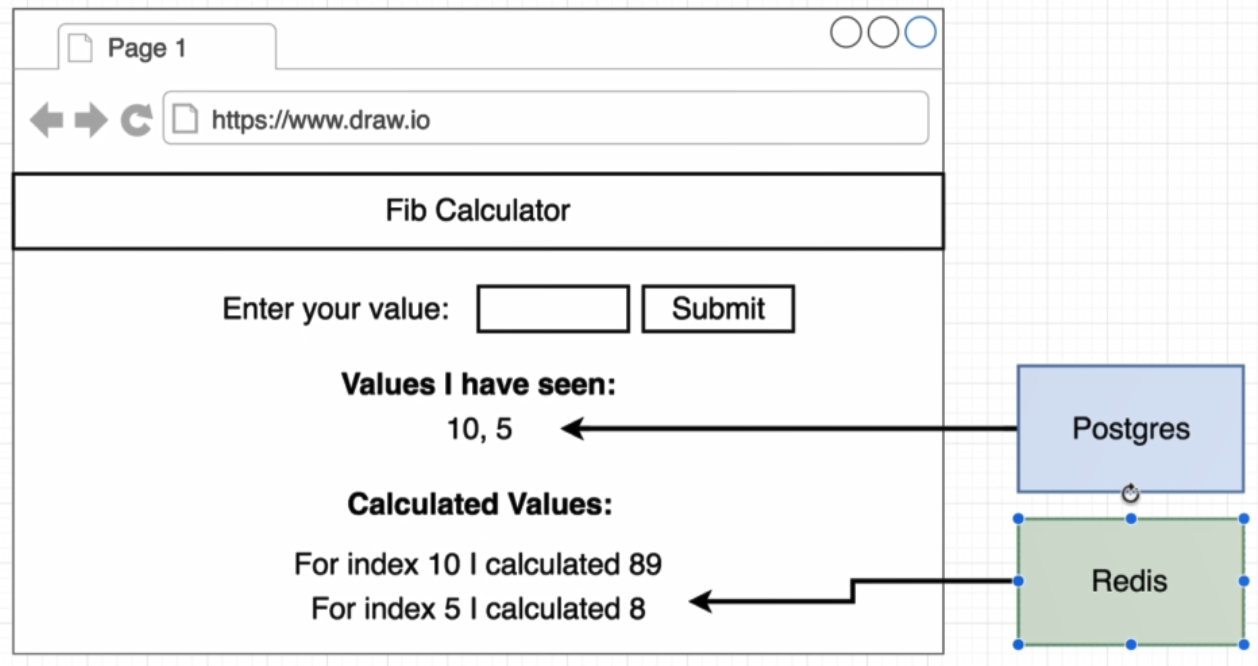


Diagram bellow show a flow of how our application is really going to behave behind the science.

When a form submit, React will request to backend Express server and it is calculate the fib value. First it is store received value into Postgre database. At the same time Express server take that index and save it inside redis database. When a new number shows in redis database, it going to wake up a separate backend node.js process(worker). The only job of worker is to watch read us for new indices that show up. When a new value shows up inside of redis the worker is going to pul that value out. It calculate a appropriate fib value for it, take that value and put it back to redis.



**Worker**

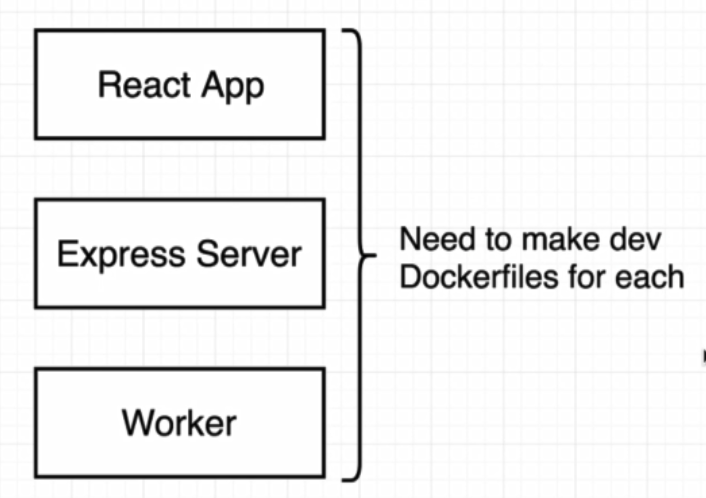
When a new index added to Redis Worker take that index and start to calculate Fibonacci result.

Create a project with name Worker. After that create index.js file to listen to redis, when an index inserted to redis, get that index and calculate Fibonacci value and put the result back to redis.

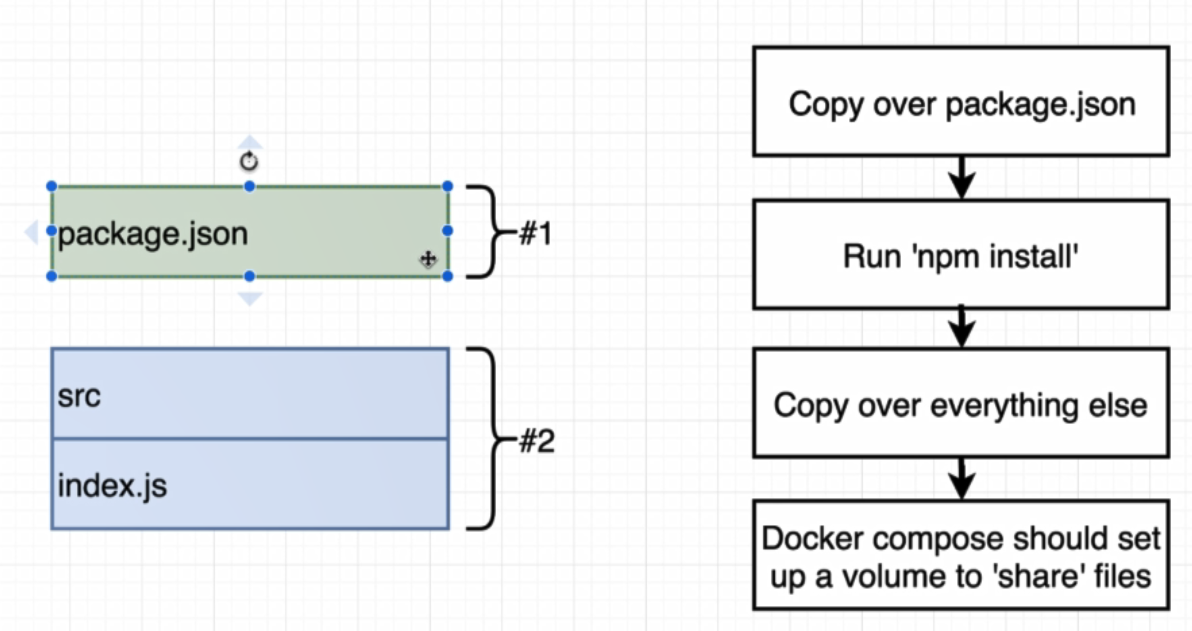
**Express Server**

It is an API layer that communicate with Redis, Postgres and React App. Beside Worker directory make an another directory with name server.

After write all the code it’s time to create Dockerfile for all the project that shown bellow:



For each one of described projects inside folder we set up a pretty similar Dockerfile workflow as bellow:



**Client:**

FROM node:alpine

WORKDIR '/app'

COPY ./package.json ./

RUN npm install

COPY . .

CMD ["npm", "run", "start"]



**Server:**

* Nodemon is a command line tool that be used to automatically reload your entire project ever any source code inside your project has changed.
* Any time we set up a volume and our source code changes, volume updates as well we get our application to automatically restart with nodemon tool.
* FROM node:alpine
* WORKDIR '/app'
* COPY ./package.json ./
* RUN npm install
* COPY . .
* CMD ["npm", "run", "dev"]



**Worker:**

FROM node:alpine

WORKDIR '/app'

COPY ./package.json ./

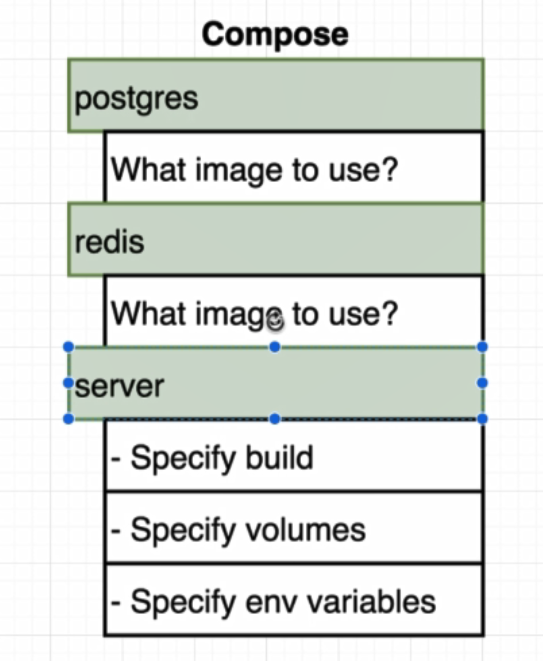
RUN npm install

COPY . .

CMD ["npm", "run", "dev"]



Now it is time to create docker-compose file.



Inside docker-compose file define options as bellow:

version: '3'

services:

postgres:

image: 'postgres:latest'

redis:

image: 'redis:latest'

api:

build:

# there is no need to specify folder (define in context:)

dockerfile: Dockerfile.dev

#use all the files inside this folder to build a image

context: ./server

volumes:

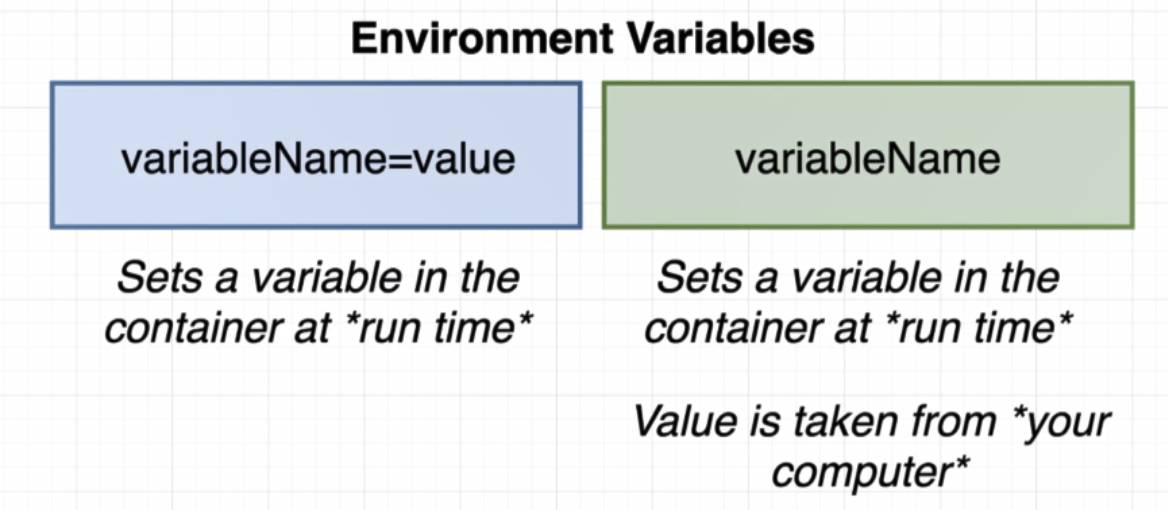
# inside app folder dont override node\_modıles folder.

- /app/node\_modules

# reference app dorectory to file and folder inside server folder

- ./server:/app

Setting up any environment variables: (used to avoid hard coded passwords)



environment:

- REDIS\_HOST=redis

- REDIS\_PORT=6379 # default port for redis

- PGUSER=postgres

- PGHOST=postgres

- PGDATABASE=postgres

- PGPASSWORD=postgres\_password

- PGPORT=5432 # default port for postgres

No we add another two services for client and worker as well:

client:

build:

dockerfile: Dockerfile.dev

context: ./client

volumes:

- /app/node\_modules

- ./client:/app

worker:

build:

dockerfile: Dockerfile.dev

context: ./worker

volumes:

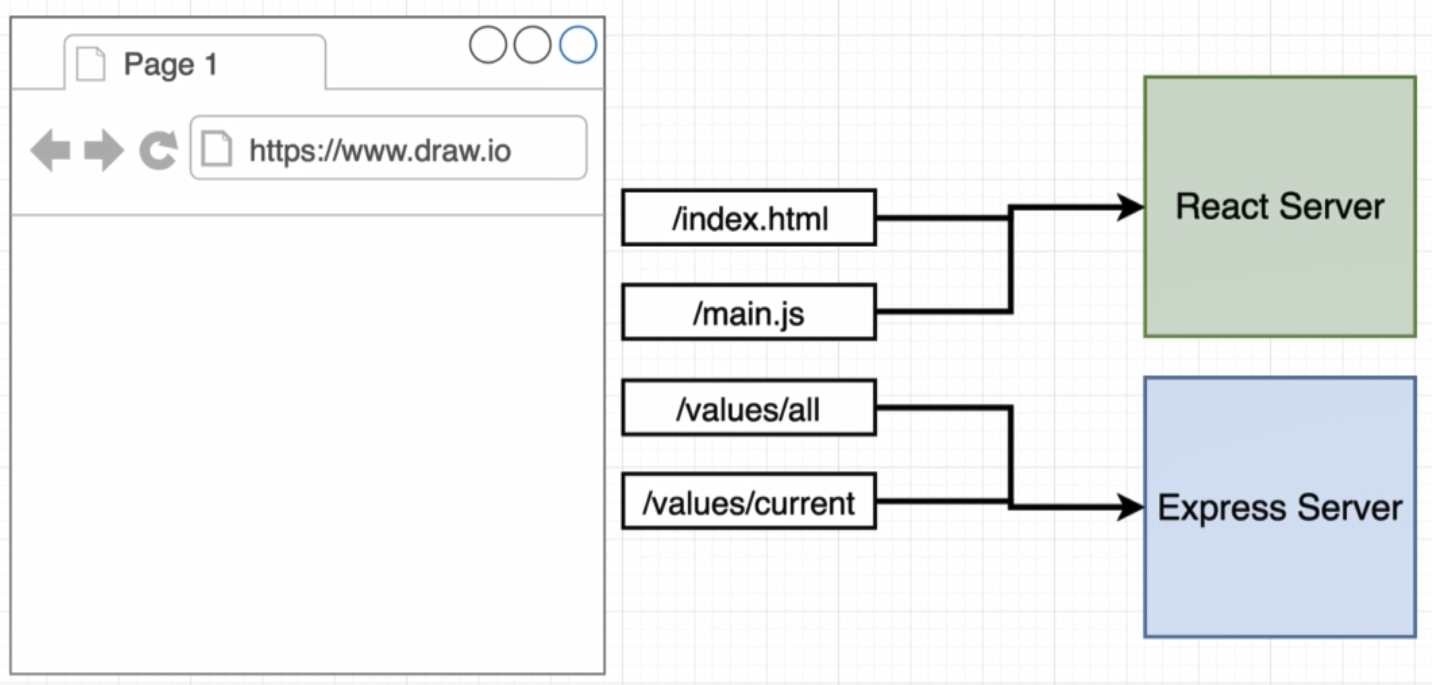
- app/node\_modules

- ./worker:/app

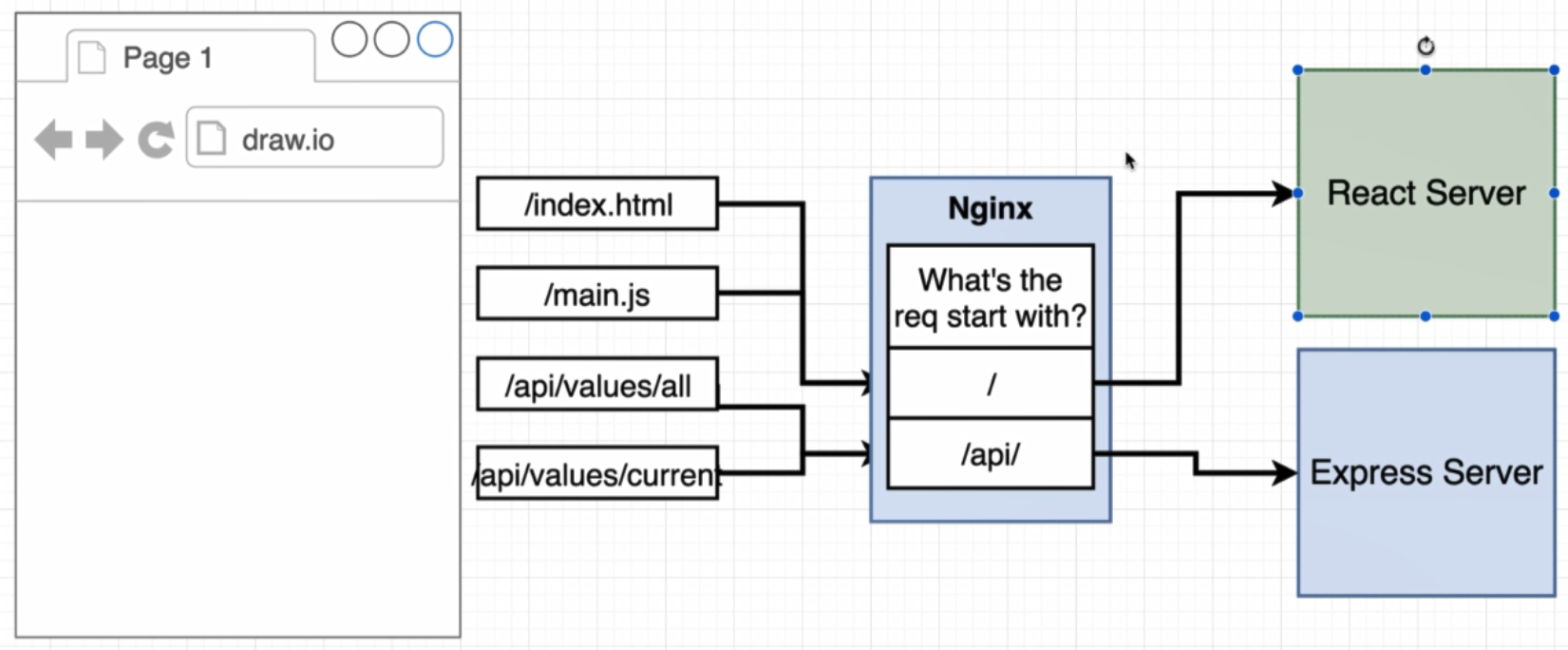
As we see on docker-compose file we made no mention of any ports inside this file. Noting exposed to the outside world. In the diagram which showed at the overall architecture of our app, we showed that Nginx is in front of everything in our project.

Why we should use Nginx in our application:

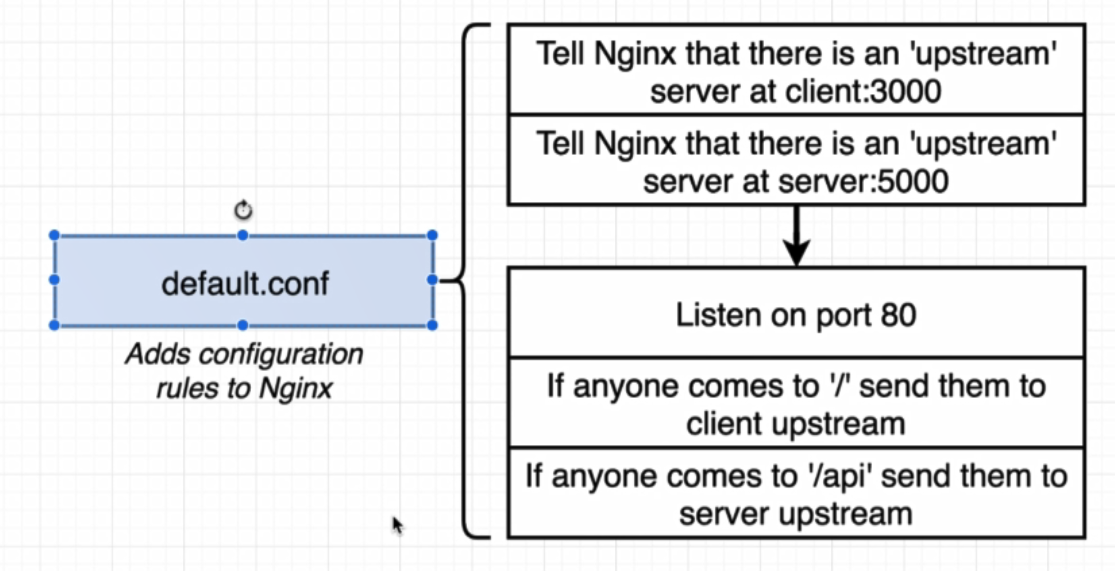
In the figure bellow /index.html and /main.js request will go through React Server and /values/all and /values/current will go to Express Server to get respond and show on broser.



To configure which request should go to which server we can use Nginx as bellow:



In order to set up Nginx in this fashion and give it a set of routing rules we are going to create a file called **default.conf**.



upstream client {

server client:3000;

}

# define server as name api (server is a keywork inside nginx)

# change server name of service inside docker-compose to api

upstream api {

server api:5000;

}

server {

listen 80;

location / {

proxy\_pass http://client;

}

location /api {

rewrite /api/(.\*) /$1 break; # rewrite url with /api to url without /api

proxy\_pass http://api;

}

}

Now we are going to create a Dockerfile that will create a new costume Nginx image and attempt to apply this configuration file to it.

FROM nginx

COPY ./default.conf /etc/nginx/conf.d/default.conf

After that add this nginx image as a service to docker-compose file.

nginx:

restart: always

build:

dockerfile: Dockerfile.dev

context: ./nginx

ports:

- '3050:80'

After docker-compuse up we have web socket error on browser. The problem is that we have not set up our nginx server to successfully allow through web socket connections. We add configuration inside **default.conf** file:

location /sockjs-node {

proxy\_pass http://client;

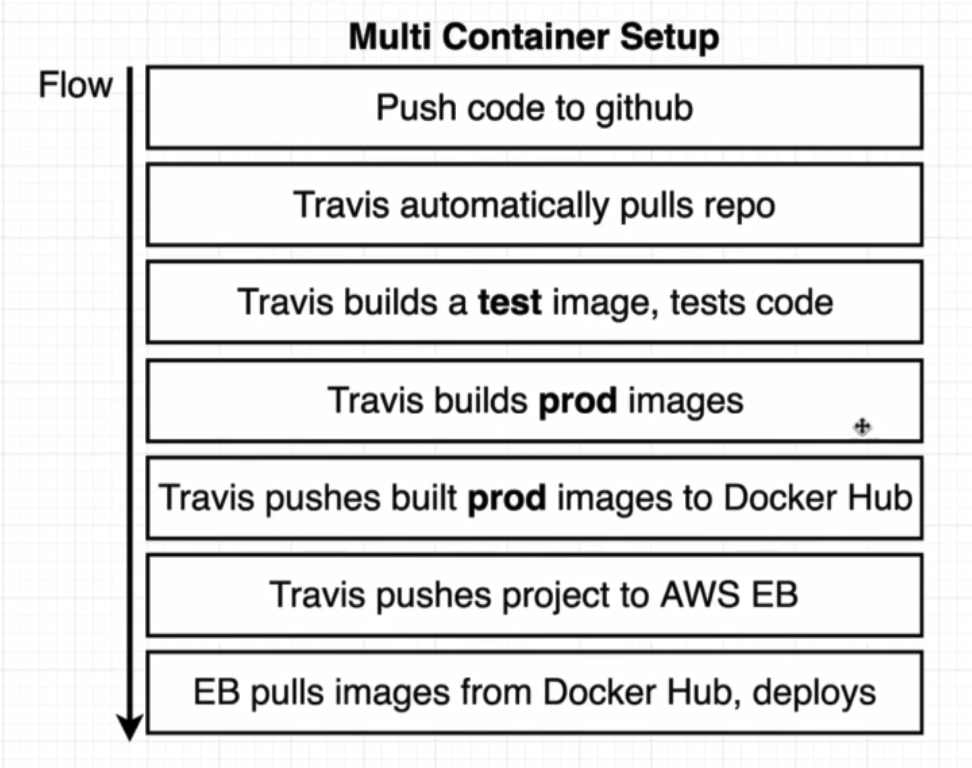
proxy\_http\_version 1.1;

proxy\_set\_header Upgrade $http\_upgrade;

procy\_set\_header Connection "Upgrade";

}

In this section we deploy our project to Amazon:



All our four project has Dockerfile.dev for development environment. Now we are going to create Dockerfile for production level.

Worker/Dockefile:

FROM node:alpine

WORKDIR '/app'

COPY ./package.json ./

RUN npm install

COPY . .

CMD ["npm", "run", "start"]

Server/Dockerfile:

FROM node:alpine

WORKDIR '/app'

COPY ./package.json ./

RUN npm install

COPY . .

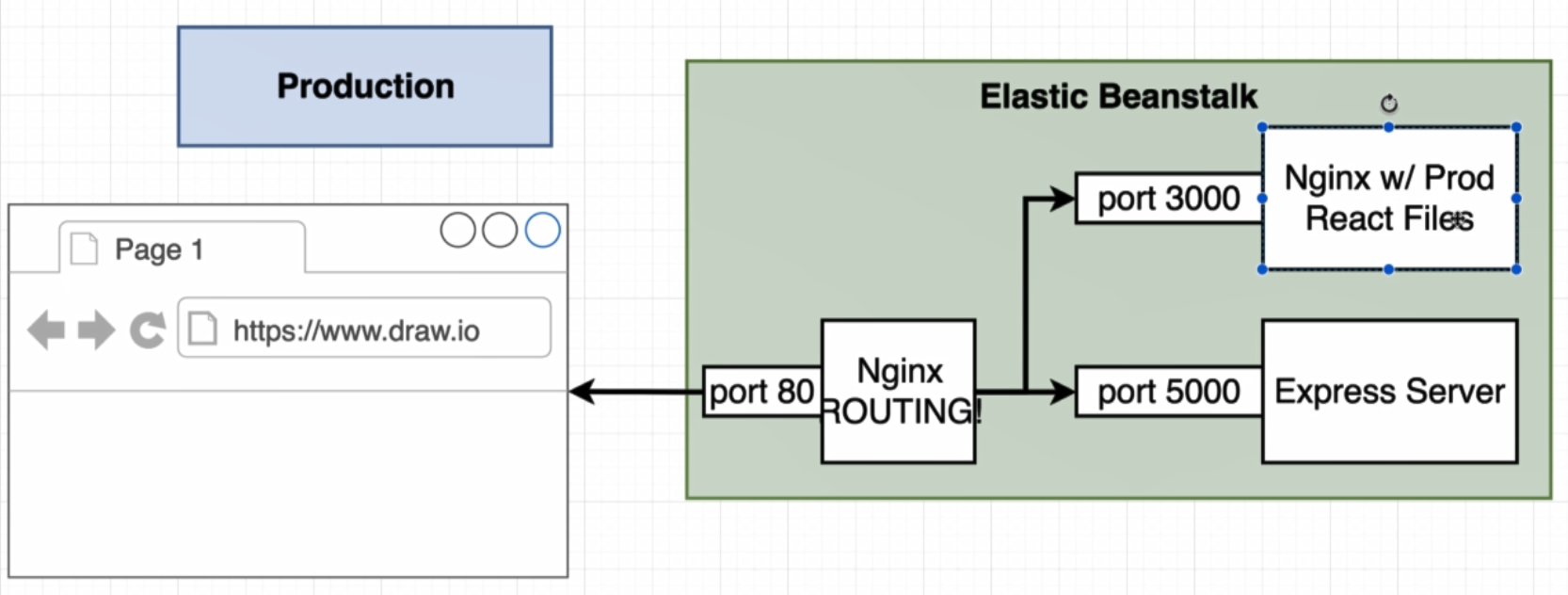
CMD ["npm", "run", "start"]

Nginx/Dockerfile:

FROM nginx

COPY ./default.conf /etc/nginx/conf.d/default.conf

Client/Dockerfile:



To create separate Nginx for client project create Nginx/default.conf inside client:

server {

listen 3000;

location / {

root /usr/share/nginx/html;

index index.html ,index.htm;

}

}

Inside Dockerfile:

FROM node:alpine as builder

WORKDIR '/app'

COPY ./package.json ./

RUN npm install

COPY . .

RUN npm run build

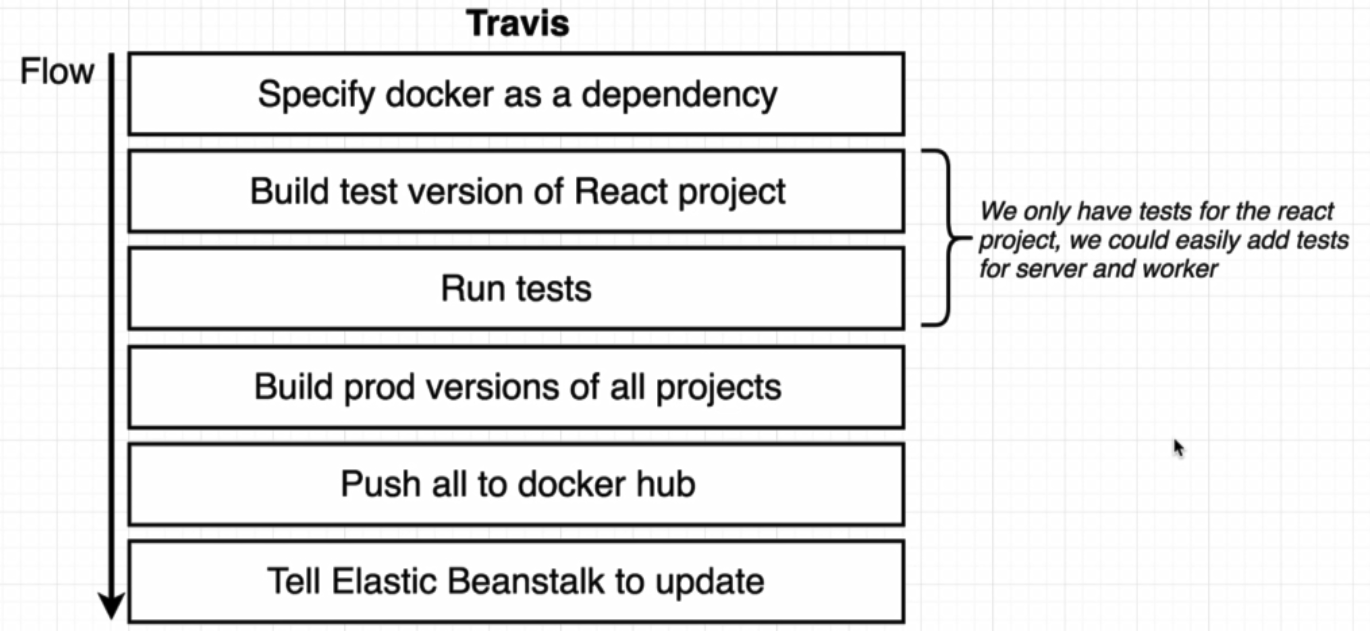
FROM nginx

EXPOSE 3000

COPY ./nginx/default.conf /etc/nginx/conf.d/default.conf

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

Create Travis file to follow step below:



sudo: required

services:

- docker

before\_install:

- docker build -t tohid1987/react-test -f ./client/Dockerfile.dev ./client

script:

- docker run tohid1987/react-test npm test -- --coverage

after\_success:

- docker build -t tohid1987/multi-client ./client

- docker build -t tohid1987/multi-nginx ./nginx

- docker build -t tohid1987/multi-server ./server

- docker build -t tohid1987/multi-worker ./worker

#login in to the docker CLI

- echo "$DOCKER\_PASSWORD" | docker login -u "$DOCKER\_ID" --password-stdin

# take those images and push them to docker hub

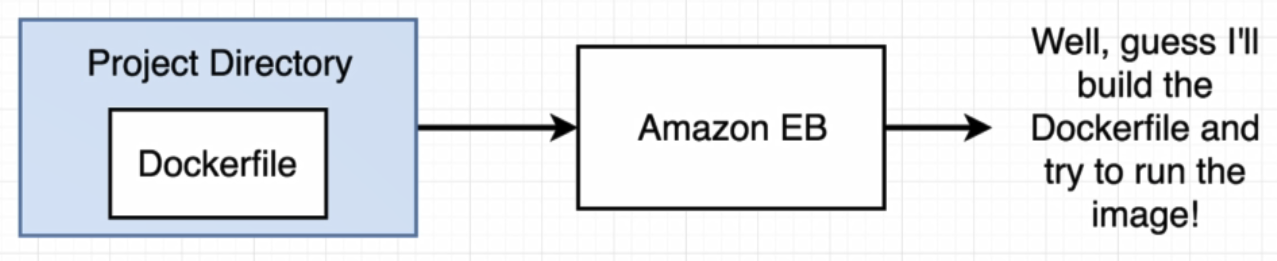
- docker push tohid1987/multi-client

- docker push tohid1987/multi-nginx

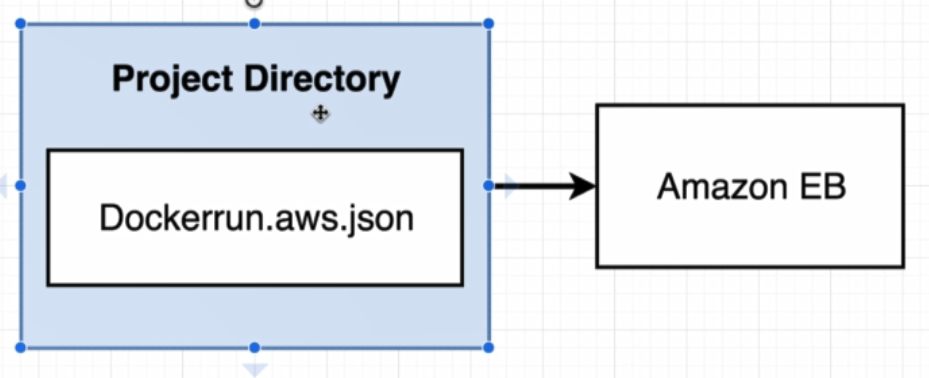
- docker push tohid1987/multi-server

- docker push tohid1987/multi-worker

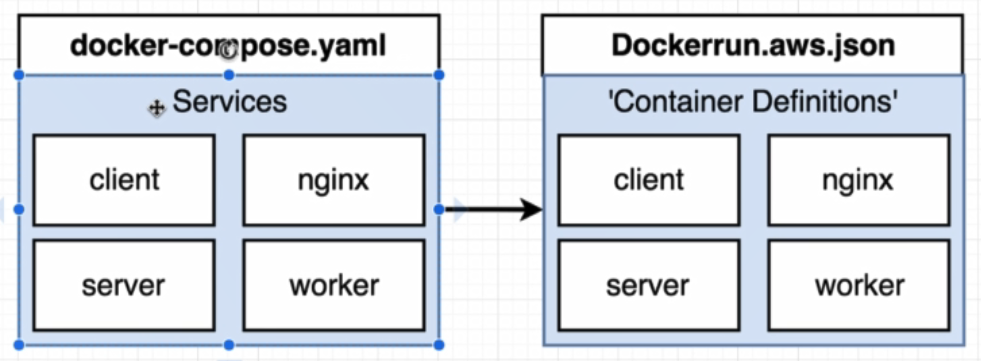
To deploy our project to Amazon Elastic Beanstalk we have to do steps below:



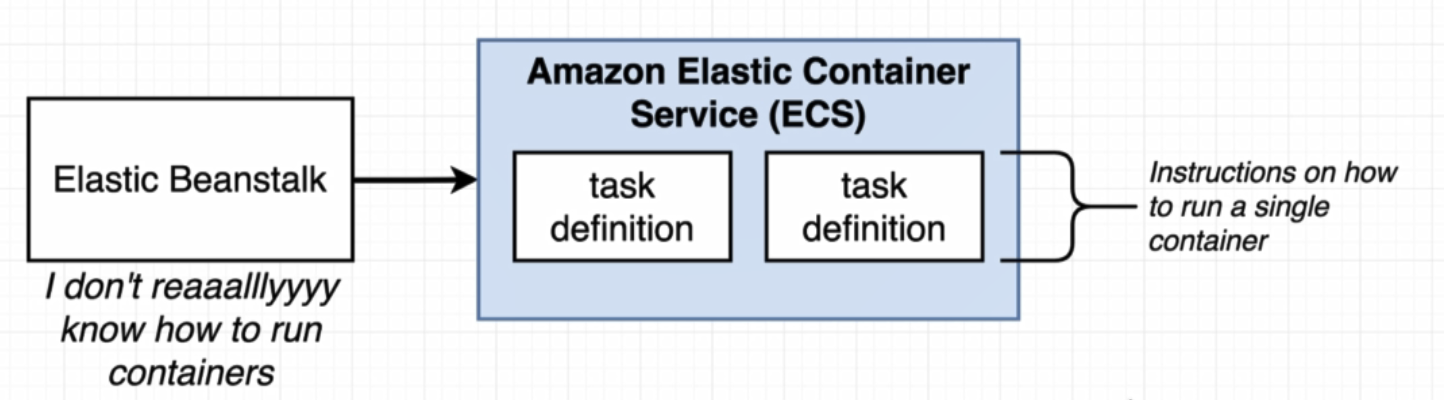
Inside project directory we create a file with name Dockerrun.aws.json



Configuration inside this file as like as docker-compose file.



Inside Amazon, Elastic beanstalk does not know how to run container:



{

"AWSEBDockerrunVersion": 2,

"containerDefinitions": [

{

"name": "client",

"image": "tohid1987/multi-client",

"hostname": "client",

"essential": false

},

{

"name": "server",

"image": "tohid1987/multi-server",

"hostname": "api",

"essential": false

},

{

"name": "worker",

"image": "tohid1987/multi-worker",

"hostname": "worker",

"essential": false

},

{

"name": "nginx",

"image": "tohid1987/multi-nginx",

"hostname": "nginx",

"essential": true,

"portMappings": [

{

"hostPort": 80,

"containerPort": 80

}

],

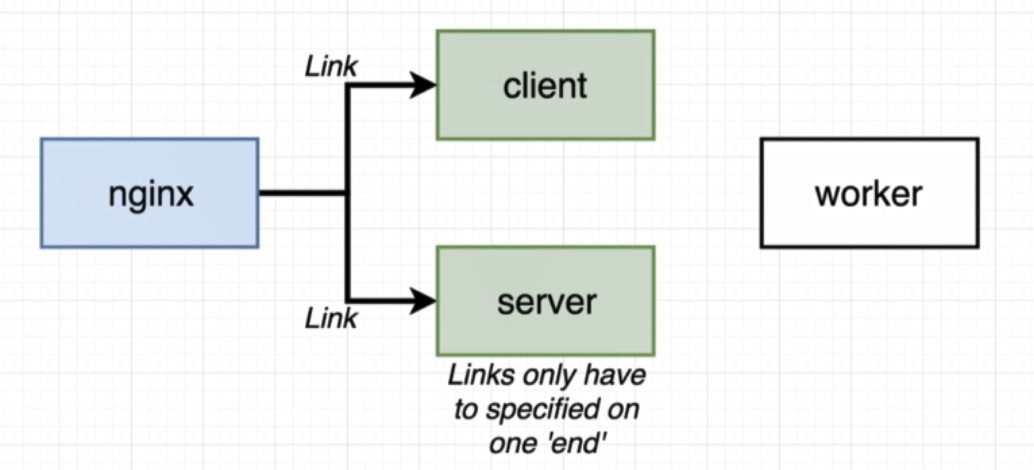
"links": ["client", "server"]

}

]

}

Links:



The diagram below display our project in production environment:

