**MULTI-DOCKER DEVELOPMENT VERSION**

-We will be using 4 docker containers namely:

i) Client

II) Server

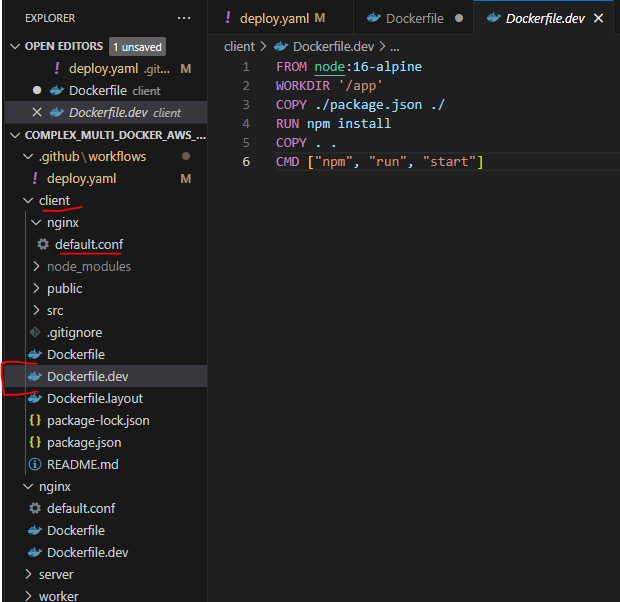
III) Worker

IV) Nginx

I) Client: have the src code for the frontend files for react,

Also additional we have the nginx server whose default.conf file specifies the information that it is listening at port 3000 and after receiving / it returns the index.html file

The Dockerfile for Client :

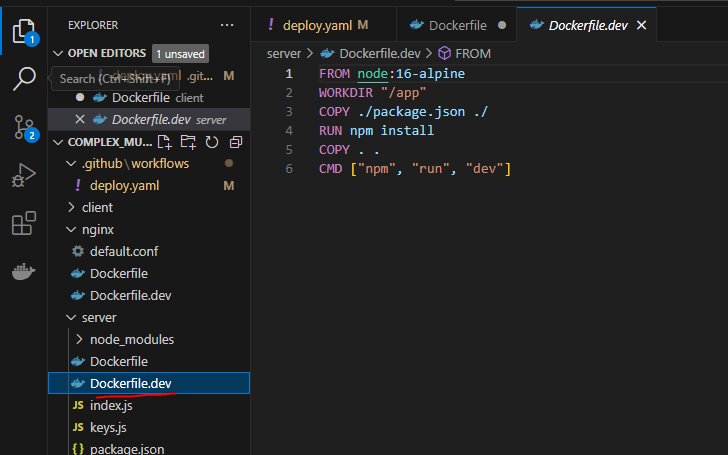


Server :

Server is called via rout /api

Server have the main file index.js which basically have the initial setup for Redis and Postgres and have the values for all the keys stored in file keys.js.

Stores the value in Postgres and worker.



Nginx :

Nginx server listen at port 80 and route the traffic to port 3000(client) or 5000(server)

A screenshot of a computer screen

Description automatically generated

Worker :

Worker performs the main calculations for the febbonic series. And sends the value on the redis.

# Build all the images command:

docker build -t multi-client -f Dockerfile.dev .

docker build -t multi-server -f Dockerfile.dev .

docker build -t multi-nginx -f Dockerfile.dev .

docker build -t multi-worker -f Dockerfile.dev .

#Run all the container on the same network:

-Create network using command :

docker network create my-network

docker run -d -p 8000:80 --network my-network --name nginx6 multi-nginx

docker run -d --network my-network --name server multi-server

docker run -d --network my-network --name client multi-client

docker run -d --network my-network --name worker multi-worker

also run the images for redis and postgres

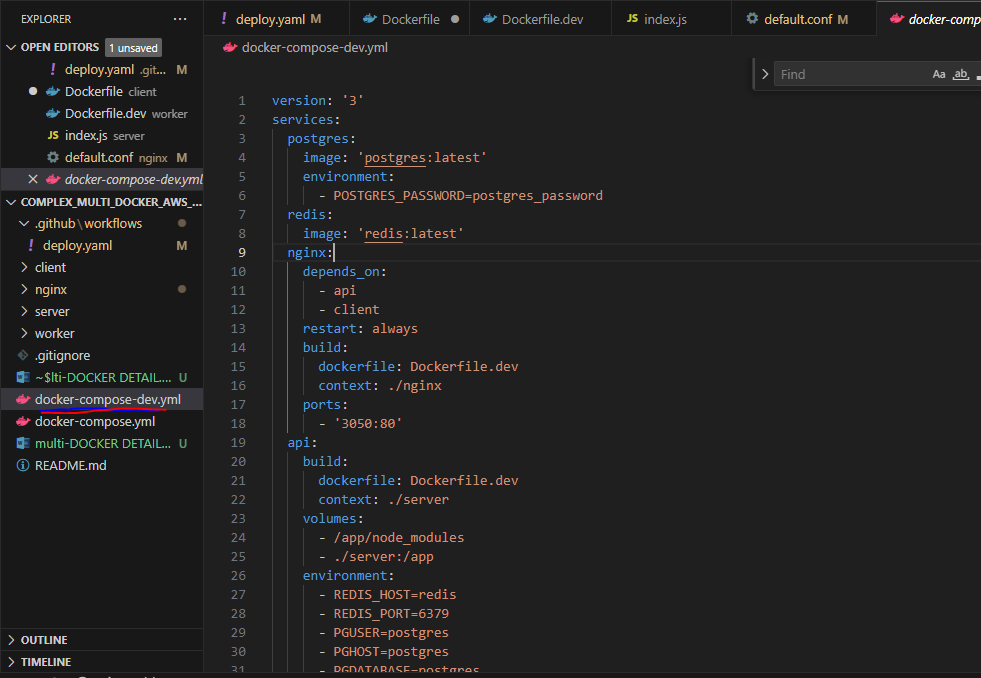
docker run -d redis

docker run -d postgres

OTHERWAY USING DOCKER-COMPOSE-DEV.YAML

* All the containers named as postgres, redis, nginx,api,client and worker are created under the same network when constructed made using docker-compose-dev.yaml

Command :: docker-compose -f docker-compose-dev.yml up



**FOR PRODUCTION**

We will use Dockerfile instead of Dockerfile.dev

A screenshot of a computer

Description automatically generated

Use of Dockerfile for use in production for client, nginx, server and worker

Deploy.yaml file for use gitActions for production and deployment on aws beanstalk:

A screen shot of a computer

Description automatically generated

Images are build and pushed to the dockerhub

And then aws beanstalk uses the zip file which make use of docker-compose.yaml

IMPORTANT : also run POSTGRES AND REDIS Instance on separate aws and put the values of environment variables.

Final after deployment:

A diagram of a computer

Description automatically generated

NOTE: AWS Elastic Beanstalk translates the Docker Compose configuration into an equivalent **Dockerrun.aws.json** file for deployment on Elastic Beanstalk.

The **Dockerrun.aws.json** file is automatically generated based on the information in your Docker Compose file. This file defines how Elastic Beanstalk should run your containers on AWS.

While you work with Docker Compose locally during development, Elastic Beanstalk uses the **Dockerrun.aws.json** file to create and manage containers in the AWS environment.